GRAHAM BELL

VIEWSHEET
VIEWSTORE

Including ViewPlot and OverView

Price
5.95
0.65
5.50
0.00
0.00
0.00
0.00
0.00

Index name

High limit
Low limit

Cost
11.90
6.50
27.00
00.00
00.00
00.00
00.00

VIEWSHEET AND VIEWSTORE

A Dabhand Guide

Graham Bell

DABS PRESS

To my father, Stanley Arthur Bell

ViewSheet and ViewStore : A Dabhand Guide

Acornsoft is a trade mark of Acorn Computers Ltd, Fulbourn Road, Cherry Hinton, Cambridge England. Within this book the letters *BBC* refer to the British Broadcasting Corporation. The terms *BBC micro, Master 128* and *Master Compact* refer to the computers manufactured by Acorn Computers Ltd under licence from the BBC. The terms *Econet* and *Tube* are registered trademarks of Acorn Computers Ltd.The names *VIEW, ViewSheet, ViewStore, ViewSpell, ViewIndex* and *ViewPlot* are trademarks of Acorn Computers. dBase is a trademark of Ashton-Tate. 1-2-3 is a trademark of the Lotus Corporation. Ultracalc is a trademark of BBC Enterprises Ltd. Any other trademarks or registered trademarks used herein are acknowledged. *ViewChart* and *Acorn User* magazine are published by Redwood Publishing, 20-26 Brunswick Place, London N1 6DJ

Disclaimer : Because neither Dabs Press nor the author have any control over the way the material in this book and accompanying programs disc is used, no warranty is given or should be implied as to the suitability of the advice or programs for any given application. No liability can be accepted for any consequential loss or damage, however caused, arising as a result of using the programs or advice in this book or the accompanying programs disc.

Contents
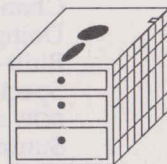
First Time Users Section

9 : Customising the Database   117

10 : Searching, Selecting, Sorting   133

11 : A Database at Work   150

12 : Diverse Reports   172

Advanced Section

13 : ViewSheet Hints and Tips   191

14 : Special Spreadsheet Functions   214

15 : Spreadsheet Integration   223

16 : ViewStore Hints and Tips   241

Appendices

Foreword

It is less than 10 years since the VisiCalc spreadsheet program changed the face of microcomputing. Small computers had until then been little more than an expensive curiosity, but VisiCalc's arrival in 1979 proved that real work could be done on a humble Apple II. Now every micro worth the name has a spreadsheet application, even the calculator-sized Psion Organiser. Given their facility for repetitive jobs, it was inevitable that micros would take on the task of filing too. It's dignified with the name 'database management', and the files are kept on floppy disc rather than in big cabinets.

These days, a credible BBC micro system can be set up at about a third of the cost of the machine back in 1982. Add VIEW, ViewSheet and ViewStore and the result is a professional business tool, yet not too complicated to be mastered either at home or in school. The Acornsoft VIEW family has been very important for the BBC micro, though it hasn't been without competitors. Computer Concepts has gathered a large base of users for it's Wordwise wordprocessor and the Inter family, and BBC Soft's UltraCalc spreadsheet is similar in capability to ViewSheet. At the time of writing, Computer Concepts has just completed its family with the release of InterBase. However, until recently only Acornsoft could offer the complete set of productivity software—wordprocessor, spreadsheet and database manager.

This book and its companion volume *VIEW: A Dabhand Guide* by Bruce Smith, aim to help you get the most from this family of software.

Acknowledgements

xi

Thanks too to my colleagues at *Acorn User* magazine, Tony Quinn and Steve Mansfield, and to all at Redwood Publishing who help make it a happy place to work. Mark Colton, author of the VIEW family, and Rob MacMillan, formerly of Acornsoft, provided useful technical information, and John Knight allowed his sideways print program to be used.

Finally, on to those who have worked on the book itself. The eye-catching cover design is by Paul Holmes, and the text has been edited and immeasurably improved by Shona McIsaac. Typesetting was done by Paul Martin at Dabs Press. All the best bits are theirs, but all the mistakes remain mine.

Graham Bell,
Belsize Park, London. August 1987

About This Book

This is the third major book from Dabs Press, a new company set up to produce books for the computer market using the latest desktop publishing techniques. The text of the book was written on a BBC model B and 6502 second processor, using VIEW and ViewSpell. The finished manuscript was transferred to an Apple Macintosh SE using BBC Soft's Modem Master and Red Ryder terminal software, and then typeset using MacAuthor. Each finished page was printed at 1.4 times its final size using an Apple LaserWriter and sent camera-ready to the printers.

Thank you...

...for buying or borrowing this book. We hope you enjoy reading it, and that you learn something from it. We welcome any comments or suggestions about the book or other Dabs Press products. Correspondence with Dabs Press or the author should be sent to the address on page ii. Alternatively, use electronic mail—our mailbox on Telecom Gold is 72:MAG11596, and on Prestel it's 942876210. *Acorn User* magazine can be contacted at Redwood Publishing, 20-26 Brunswick Place, London N1 6DJ, or on Telecom Gold, 81:RED001. Letters can't all be answered personally, but we will try. All correspondents and mail-order purchasers of this book will be advised automatically of further Dabs Press products, unless they request otherwise. Personal details held will of course be provided on request, in accordance with the Data Protection Act.

1 : An Introduction to the View Family
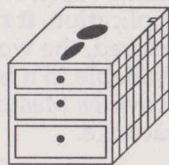
The VIEW Family

When a new microcomputer is launched, its success depends upon the software which goes with it. In designing the original BBC model A microcomputer, attention was paid to the needs of software other than BASIC programs and the inevitable video games, so that the machine would be suitable for as wide a range of applications as possible. Perhaps the most obvious outward sign of this was the inclusion of a TAB key on the keyboard: there is little need for it except in wordprocessing. The first extra sideways ROM released for the new machine was a wordprocessor. Although not the first, Acornsoft's VIEW wordprocessor in its original form was one of the most popular sideways ROMs for the BBC micro.

The first VIEW was joined by ViewSheet, a spreadsheet, and then ViewStore, a powerful but complex database manager. The wordprocessor, spreadsheet and database manager form the working core of any business software family.

Acornsoft has since released several revised versions of VIEW: 2.1 and the current standard 3.0 (versions of this are included with the Master series micros). Other recent additions to the family include ViewIndex, an indexing program, ViewSpell, a spelling checker, and ViewPlot which enables graphs and diagrams to be prepared.

The popularity of the VIEW family has been enhanced by the inclusion of both VIEW and ViewSheet as standard in the BBC Master series micros. However, the UK version of the Master Compact contains only the wordprocessor. The others can be added to any BBC micro, by plugging in the appropriate sideways ROMs. Disc versions of ViewIndex and ViewPlot are also available and some of the family are still sold for the Acorn Electron, in cartridge form.

An important new development for the Master series micro is OverView. As its name suggests, this is a kind of family manager; it

allows rapid switching between each VIEW application. On leaving one application, it retains the current work. Next time that application is entered, the work formerly in progress is restored. OverView is only available in a Master cartridge, or on disc for the Master Compact, but provides Master series owners with the whole VIEW family in a single package.

This use of software from the same family is better than using programs from several different sources for two reasons. First, they all look similar on screen, have many commands in common, and use some function keys in the same way. Learning to use the applications is therefore simpler. Second, it is often easier to exchange data between members of the same family than between two unrelated applications.

Without OverView, the sharing of data is not really a strong feature of the VIEW family. Transfers can't be made directly between modules, or by sharing files in the manner used by more integrated programs, but there are ways that material can be moved between VIEW and ViewStore, from ViewStore to ViewSheet and from ViewSheet to VIEW. All three of these packages can be used to provide data for ViewPlot. These data sharing abilities are summarised in figure 1.1
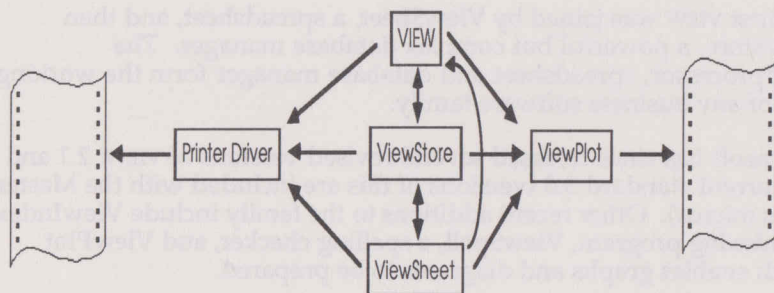
Figure 1.1. OverView Data Sharing

This book provides the novice with an introductory guide to the use of ViewSheet and ViewStore. Later chapters give experienced users an insight into more complex features. ViewPlot is also explained.

All details relating to ViewStore and ViewSheet are also applicable to OverView on the Master series micros and additional OverView features are discussed.

The ViewSheet Spreadsheet

Spreadsheets like ViewSheet are useful for repetitive numerical work. In business, this includes costing exercises, sales analyses or forecasts, and financial planning. At home, budgeting can be made easier. A spreadsheet can be used to work out interest problems with mortgages and loans. As spreadsheets are common business tools, learning how to use them is a valuable exercise at school. It can also be used like a calculator. When done by hand, statistical work is notoriously repetitive, but a spreadsheet makes life much simpler. It helps to collate numerical information, and can be used to derive many statistical descriptions like mean and standard deviation.

All these types of exercise can be done on paper, and with the help of a calculator they might not seem too daunting. So why use a spreadsheet? If a paper calculation is worked out for mortgage repayments, what do you do when the interest rate changes? If a mistake is discovered in the data used to work out a standard deviation, what do you do? How can the effect of a change in the price of raw materials upon overall product costs be investigated? This means working out all your calculations again – a very time-consuming task. With a spreadsheet, all you do is press a few buttons, and the answer is there in seconds.

This feature makes the spreadsheet the powerful business tool it is. It allows experimentation with figures. How much would reducing stock levels in the slack months save? Alter a few figures, and the revised costing appears instantly.

The ViewStore Database Manager

A database is a store of organised information: domestic examples include card indexes and address books. At its simplest level, ViewStore can be used to keep the electronic equivalent of a card index. The information on the electronic cards can be almost anything: a telephone list; a set of references to magazine articles; a list of books, stock parts or business transactions with information on each item. Any particular ViewStore card can be found very quickly, and the information shown on screen or printed out.

Unlike a traditional database, the electronic cards don't have to be kept in one set order. A telephone directory is sorted in order of surname, so it can only be used by knowing the name of the person. Computer-based data can be sorted into several different orders, so a person's name could be found by knowing only their telephone number or their address.

Complex questions can be answered quickly from an electronic database. How many people on the list live in Milton Keynes? Is there a reference to the word 'granite' in the title of any books in the library? What is the total wholesale value of the stock of windscreen wipers? Searching for this type of information is often too time-consuming to be worthwhile with card indexes. The computer makes the same information easily accessible.

There is a penalty to pay; the electronic database is more complex than a simple index or phone book. Thought has to be put into the database design before anything is typed in. There is no electronic equivalent of scribbling extra notes on the corner of a traditional piece of card!

Graphics with ViewPlot

ViewPlot is intimately related to both the database manager and the spreadsheet modules of the VIEW family. It is a charting utility to present numerical data and results from both of the other packages in graphical form.

OverView Goodies

OverView provides a convenient and cost-effective way of adding the remainder of the VIEW family to the applications already provided with the Master 128. In addition to the usual features, OverView adds a few new commands to the Master series. For example, you can have a 106–character wide screen, which is a boon on large spreadsheets.

Other commands simplify data transfer into ViewSheet, and allow temporary switching from one language to another without having to keep saving and reloading time and time again. On-screen help for the whole family is also included.

Compatibility

The members of the VIEW family work with all BBC micros fitted with operating system (OS) version 1.2 or later. They are compatible with the various Acorn Disc Filing Systems (DFS), and with the Advanced Disc Filing System (ADFS). The Econet network, the Network Filing System (NFS) and Advanced Network Filing System (ANFS) are also compatible. OverView requires a Master 128 micro.

A fast filing system is necessary to get the best out of any of these applications. ViewSheet can also use cassettes for file storage, but using cassettes makes loading and saving slow and some spreadsheet features can't be used.

Using ViewStore or ViewPlot with DFS 0.9 isn't recommended, as it can result in the corruption of the disc catalogue. Fitting DFS 1.2 or a later version is required. If using ViewStore on a network, an upgrade to ANFS speeds up operations.

Type Conventions Used

In this book, various signs are used in the following way :

KEY	This denotes a single key on the keyboard, for example ESCAPE. UP is the grey up-cursor key (also DOWN, LEFT and RIGHT). Similarly, f0 is the first red function key. Function keys may also be indicated by the relevant command; these are marked on the narrow black function key strips. RETURN is used to indicate pressing the return key, but only where it is unexpected. It often needs to be pressed at the end of a line of input even where not marked.
CTRL–@	All keys can be used in conjunction with SHIFT and CTRL, for example CTRL–@ or SHIFT–COPY. This means hold down the SHIFT key, briefly press the COPY key and then release the SHIFT key.
<file1>	Means 'file1' must be replaced by something appropriate according to the context, eg, a file name.

VAV—B

[200]   Shows '200' is optional (this also appears as [<file2>], which means that an appropriate filename is optional).

Bytes free This typeface distinguishes computer text, often something that is displayed on screen or printed out.

Mode 3 Denotes some text that should be typed at the keyboard, usually a command.

Throughout this book, some familiarity with a BBC micro is assumed, but no previous knowledge of ViewStore, ViewSheet or ViewPlot is needed. Experience of the VIEW wordprocessor is an advantage, but not absolutely necessary.

2 : Beginning With Viewsheet
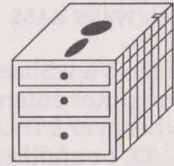
Spreadsheet software has been an important driving force in the development of the microcomputer market. The first micro-based spreadsheet, Visicorp's Visicalc, enabled the original Apple II computer to become the first real business machine, though it had less impact in Europe than in North America. Similarly, the release of Lotus 1–2–3 added extra impetus to the bandwagon after the introduction of the IBM PC in 1981. Although Lotus 1–2–3 is an integrated package covering database management and business graphics, its huge popularity is founded upon the capability of its spreadsheet.

ViewSheet has not had such a catalytic role to play in the BBC micro market. The Acorn machines were already well established before its release, especially in the educational sector. However, the development of spreadsheet applications such as ViewSheet, has been vital in attracting business users to the BBC micro.

ViewSheet is a typical general purpose spreadsheet for a micro, and compares favourably with others used on the BBC micro. The principles learned using ViewSheet can be easily applied to other spreadsheet programs for the BBC micro and other computers.

A First Look

This section introduces the fundamental features of ViewSheet, and describes the construction of a simple spreadsheet.

Starting Up ViewSheet

The ViewSheet application is normally available as a 28-pin ROM to be installed in one of the sideways ROM sockets in the computer. Simple instructions for doing this come with the ROM. The key strip should also be put above the function keys. With the Master 128, fitting a ROM is unnecessary, as ViewSheet is already installed within the Megabit ROM. Licences are also available from Acorn to allow one copy of

7

ViewSheet to be used on all the micros attached to a network, using sideways RAM.

If the ViewSheet ROM is inserted in the highest priority socket, the computer enters the application immediately upon switching on, or whenever CTRL–BREAK is pressed. More usually the computer is in BASIC or another language, and ViewSheet can be started by typing:

```
*SHEET
```

at the usual '>' or '=>' prompt.

At this stage, a familiar VIEW family screen is displayed: it shows the application name, the amount of memory free and the screen mode in use. This is the *Command screen*:

```
ViewSheet

Bytes Free 24846
Editing no file
Screen Mode 7
Printer default
=>_
```

The flashing cursor lies beside the Command prompt '=>', and various commands can be entered here. Many of these duplicate commands used in VIEW. As an example, type:

```
MODE 3
```

to change the screen mode. All the usual system commands can also be used, to catalogue discs (*CAT or *.), to define the function keys (*KEY) and to switch to another language (for example, *BASIC).

ViewSheet works in a similar way to VIEW; pressing ESCAPE at the Command prompt makes the spreadsheet appear. This is the *Sheet screen*. Pressing ESCAPE again at any time returns to the Command screen. Commands also work in a similar fashion to VIEW; command words like MODE 3 work only in Command mode. On the Sheet screen single key commands are assigned to each of the function keys, such as GO TO SLOT (f7) or INSERT CHARACTER (f8).

This separation of Command and Sheet modes is perhaps the least typical feature of ViewSheet. Most other spreadsheets allow commands to be entered onto the Sheet display, usually by prefacing them with a '/' or '\' character, or by using a Command menu.

At the very top of the screen is ViewSheet's Sheet display, a four-line area set aside for status information. Most of the Sheet screen is made up of a large number of boxes or *cells*, sometimes also called *slots*. The cells are arranged in horizontal *rows* and vertical *columns*. Across the top of the cells is a line of dots showing the names of several columns, and running down the left is a dotted border and the names of some of the rows. Column names are designated by letters, and the rows are referred to by number. Any single cell can be referred to by its column and row names, column first. For example, the top left cell is named A1. Below this is cell A2, and to the right of A2 is cell B2. Cell C4 is highlighted in figure 2.1.

```
   A    SLOT=A1
        CONTENTS=*BLANK*


   0       .......A.......B.......C.......D.......E
   ......1>
   ......2
   ......3
   ......4                      CELL C4
   ......5
   ......6
   ......7
   ......8
```

Figure 2.1. The Sheet screen, illustrating cell C4.

The Active Cell

One cell on the screen is picked out in white, and with a '>' prompt. This is the *cell cursor*, and it marks the *active cell*. The name of this active cell is shown at the top of the screen, for example, 'SLOT=A1'. In mode 7, the active cell is not picked out in white, and is instead indicated only by the '>' prompt.

The active cell can be changed with the cursor keys. Press the DOWN key once to make A2 the active cell, and RIGHT to make B2 active. SHIFT combined with the cursor keys makes the active cell move by a whole screen width instead of one row or column at a time. If you move the active cell as far as possible in each direction, you'll discover that the whole sheet is 255 rows deep (labelled one to 255) and 255 columns wide (labelled A to Z, AA to AZ, BA to BZ and so on, to IU).

Another way to change the active cell is provided by the GO TO SLOT function, by pressing f7 followed by the name of the required cell. For example typing:

f7 A1

makes A1 the active cell.

The whole sheet contains about 65,000 cells, plenty of room for even the largest jobs that the BBC micro can handle. Only a few of these cells are visible at any one time, the exact number depending upon which screen mode used. If mode 0 is used, nine columns and 26 rows are shown, but in other modes, fewer cells are shown on screen. No matter which mode is used, the Sheet screen can be scrolled to bring any of the 65,000 cells into view.

The Contents of Cells

On a new sheet, each cell starts off blank and this is shown at the top of the screen, for example: 'CONTENTS=*BLANK*'. A cell can be filled with one of three types of item: a *value*, which is a piece of numerical data such as '127', a *formula* like 'A5 * 10 + A6', which describes how the data from other cells should be used to get the desired results, or a *label*, which is text such as 'Total to date'.

Entering Labels

The simplest cell type is a label. To put a label into a particular cell, perhaps B3, first make that cell active, by using the cursor keys or GO TO SLOT. Once the cell B3 is picked out in white, the label can be entered by typing in a word. Characters typed don't appear directly in the active cell, but at the top of the screen next to a new flashing cursor that appears as the label is typed in. This is the *Edit line* and *Edit line cursor*. A label might be the name of a month such as 'JANUARY'. The label is transferred from the Edit line to the cell when RETURN is pressed. At the top-left of the screen, the 'L' shows that the cell now contains a label. A label can be anything that is not either a value or a formula.

Editing the Cell Contents

If you type in the wrong thing the contents of a cell can be corrected. When the cell containing 'JANUARY' is active, it can be changed simply by typing in a new label:

    MARCH

This is the quickest way if the change is a major one.

If the error is small, the cell can be modified. Pressing the COPY key copies the current cell contents to the Edit line, where the text can be altered. To change a single character, simply type the new character whilst the flashing cursor is over it. The flashing cursor can be moved along the Edit line by using CTRL–LEFT and CTRL–RIGHT.

To make bigger changes, some of the function keys can be used just as they are in VIEW. Characters to the left of the cursor can be replaced with spaces by pressing DELETE. The character at the cursor is removed with f9. Pressing f8 inserts a space into the line at the position of the Edit cursor. DELETE END OF LINE (key f3) deletes the part of the line beyond the flashing cursor. Pressing f4 or f5 moves the Edit cursor to the start or end of the line respectively. After the correction has been made, RETURN inserts the new contents back into the active cell. Whilst using the Edit line, ESCAPE abandons the alterations and the original contents of the cell are retained. Practice all this by changing 'MARCH' to 'APRIL'.

If RETURN is pressed and ViewSheet decides there must be an error in the line, the computer beeps. When editing that cell's contents, 'Error?' is displayed in the cell and a message may be shown in the status area; the cell contents can be copied to the Edit line with COPY and then corrected. At other times, the flashing cursor is automatically placed on the offending text, and an appropriate error message is displayed. The line can be re-edited and RETURN pressed again.

Labels can be of any length up to 240 characters, because that is the maximum size of the Edit line. Of course neither the Edit line nor the cells can display all of a label that size. Cells 10 units wide, for example, always show the first 10 characters of the label. The Edit line can be scrolled left or right using CTRL–LEFT and CTRL–RIGHT, or the BEGINNING OF LINE and END OF LINE function keys.

Entering Values

To enter a value in a particular cell, perhaps next to 'MARCH' in cell C3, first make that cell active. Nothing can be typed into any cell until the cell cursor is moved to that cell. Now type a number, say 59, in the Edit line, and press RETURN. The number is transferred to the cell and the flag at the top left becomes 'v' to show the cell contains a value.

Value cells can only contain numbers: either integer numbers like 1 or 6 or −15, or real numbers like 0.75, −12.1 or 3.921569. Unlike BASIC, ViewSheet treats integers and real numbers in exactly the same way. Really big or very small numbers can be expressed in exponent notation: 1.23E6 means 1.23 multiplied by 10 to the power of six (1.23×10^6), which is 1,230,000.

The numbers entered into cells are often called *constants*; although they can easily be changed by editing the slot contents, they can't vary while a sheet is recalculated. Now the following labels and constants can be entered into the cells below 'APRIL' and '59':

```
MAY       4 6
JUNE      4 5
```

The value 45 should go in cell C5, and the sheet should now look like figure 2.2. If there are any cells filled that should be empty, they can be emptied using SHIFT–f9. This is a DELETE SLOT function, and it returns a cell to its original blank state. If a cell contains something wrong, typing in new information or editing the cell contents works in exactly the same way for values as it does for labels.

```
VA   SLOT=C5
     CONTENTS=45

0    ......A......B......C......D......E
......1
......2
......3          APRIL       59
......4          MAY         46
......5          JUNE    >       45
......6
......7
......8
```

Figure 2.2. Trial sheet with three labels and three values.

Formulae

A formula is like an expression in BASIC. It is an instruction to do a numerical calculation. This expression or formula might contain constants in the same way as value cells, use arithmetic operations like addition or division, call mathematical functions like SIN or LOG or use a few special spreadsheet functions like ROW.

In BASIC, an expression may also make use of variables - in ViewSheet, the equivalent is a *cell reference*. A reference is the name of a cell, and the numerical calculation makes use of the value in the named cell, just as a BASIC expression makes use of the value of a variable.

To enter a formula into cell C7, make C7 the active cell. Now type:

```
C3 + C4 + C5
```

in the Edit line. This is the formula, and it remains associated with the cell; 'CONTENTS=C3+C4+C5' is shown in the status area. However, what is shown on the sheet itself is the value of the formula, 150, because 59 + 46 + 45 = 150. The cell contains the formula, but the sheet shows only the value of the formula. To emphasise this, the flag at top left shows 'v'.

It is good practice to mark any formulae to show what they do. The formula in C7 adds together the three values above it, so put the label 'TOTAL' in cell B7.

Adding up a column of figures is such a fundamental operation for a spreadsheet that there is a shorthand way of expressing 'C3 + C4 + C5'. This can be used in the next formula, to work out the average of the three values. Put this into cell C8:

```
C3 C5 / 3
```

Now the status area shows 'CONTENTS=C3C5/3'. In this formula, C3, C5 is a *range*, and it indicates the sum of the values in the range of cells between C3 and C5. The average of the three numbers is this sum divided by three, and the value of the mean (50) is shown in the cell.

The sheet should now look like figure 2.3. The sheet has all three types of cell on it: labels, values and the two new formulae.

```
VA   SLOT=C8
CONTENTS=C3C5/3

0       .......A.......B.......C.......D.......E
......1
......2
......3          APRIL       59
......4          MAY         46
......5          JUNE        45
......6
......7          TOTAL      150
......8          AVERAGE >    50
```

Figure 2.3. The trial sheet after defining two formulae.

But Why Use a Spreadsheet?

A computer is useful if it helps in doing something better, quicker or more cheaply, or does something new. A spreadsheet fails the last, because everything that a spreadsheet application can do, can also be done by hand, with a pencil and paper, or with the help of a calculator. So how can ViewSheet help?

Try this: move the cell cursor to C3 and replace 59 with any other number, say 68. When you press RETURN, 59 is overwritten by the new number, but the total in cell C7 and the mean in C8 both change too! This is why a spreadsheet is useful. Once the formulae are set up, they are recalculated automatically if any changes or alterations are made to the data values.

Imagine the spreadsheet is being used to calculate the cost of a new garden wall. The cost of the wall is made up of a number of smaller costs: the price of the bricks to build the foundations and wall, cement and sand for mortar, capping stones for the top of the wall, and labour. Certain costs depend on the length of the wall, other costs depend on the height as well as the length. Instead of doing each calculation individually for every job, a builder might construct a spreadsheet that allows him to quote a price for any length and height of garden wall, including the right formulae for each element of the total cost. Just type in the length and height needed and the price is calculated immediately.

This rapid recalculation is also invaluable when a spreadsheet is used for forecasting, perhaps a business plan for some new venture. The spreadsheet can be used to answer 'what if' questions. What will happen if the interest rate goes up? What if sales are only 300 per

month? Alter the cells containing the rate or the sales figures and the forecasted financial position will be updated immediately. Experimenting with the data on the sheet can help to make the financial position understandable.

Go Forth and Replicate

Other reasons for using a spreadsheet can be demonstrated by adding the following labels and data to the sheet:

cell B1	YEAR
cell C1	This RETURN SHIFT–f8
cell D1	Last RETURN SHIFT–f8
cell D3	53
cell D4	41
cell D5	38

Pressing SHIFT–f8 (the JUSTIFY LABEL function) helps to tidy up the sheet by altering the way the label is printed in the cell. The label is aligned with the right edge of the slot rather than the left edge. A further press of SHIFT–f8 changes it back again.

Doing all this makes the sheet look like a comparison of two sets of data. It would be useful to compare the total and average figures too. Of course you could type in the formulae again, into cells D7 and D8, but there is a better way.

The REPLICATE function is a means of copying the contents of cells to other cells; the contents may be labels, values, or as in this case, formulae. To copy the formula for the total from C7 into D7, press f0. The prompt 'From - To?' appears in the status area, so type in the source and destination cells, separating them with a hyphen:

```
From - To?
C7 - D7
```

If the source cell contained only a label or value, it would be copied straight away, but C7 contains a formula C3 + C4 + C5. This formula contains cell references, so there is a problem: should the new, copied formula refer to the same cells, or to different cells?

If the formula were copied without change, then the value in cell D7 would be the same as in C7. This would be an *absolute* replication,

15

because the formula would always refer to cells C3, C4 and C5 no matter where the formula was on the sheet. However, this formula should not refer to specific cells, but to cells with the same position relative to the formula. Think of this as *in the column above*, rather than *in column C*. This is a *relative* replication, and so C3 should be altered to D3, C4 to D4, and so on. Now answer the prompt for each cell reference in the formula:

```
R)elative, N)o change?
R
```

As this is done, the formula is shown in the status area, with each cell reference highlighted in turn. In this case, R must be pressed three times, as there are three cell references in the formula being replicated. Pressing N instead of R would result in an absolute replication.

Replication is a powerful tool. It can deal with ranges of cells, large blocks of cells, and mixtures of absolute and relative references. These will be introduced later.

An Important Point

Begin to replicate the average formula in the same way, by pressing f0. When the 'From - To?' prompt appears, move the white cell cursor to C8 with the cursor keys, and press SHIFT–COPY. This copies the name of the cursor's cell onto the Edit line, where it forms part of the formula. Now type a hyphen, move the cell cursor to D8 and use SHIFT–COPY again, then RETURN to end the line. While *pointing* to the cells C8 and D8 with the white cursor, think of this as replicating from *there* to *there*.

Once again, this is a relative replication, because the D8 formula should refer to the cells above D8, not the actual cells used in the C8 formula: C3 C5 / 3 should be changed to D3 D5 / 3. The whole replication sequence looks like this:

```
f0
From - To?
C8 - D8
R)elative, N)o change?
R R
```

Note replication does not necessarily involve the white cell cursor. It is only used for pointing.

Pointing is equally useful when entering formulae. When the flashing Edit line cursor has appeared, the cell cursor can be used for pointing. When RETURN is pressed, the cell cursor snaps back to the original active cell, and the completed formula is put there. Pointing is generally less prone to error than naming each cell individually.

```
VA   SLOT=D8
CONTENTS=D3D5/3

0     .......A.......B.......C.......D.......E
......1          YEAR     This   Last
......2
......3          APRIL     68     53
......4          MAY       46     41
......5          JUNE      45     38
......6
......7          TOTAL    159    132
......8          AVERAGE   53>    44
```

Figure 2.4. The trial sheet almost complete.

Saving and Loading Whole Spreadsheets

This first *model* is now almost complete. A complete sheet such as this is often called a 'model' because financial planning and modelling have been the major use for spreadsheets. To conclude this brief look at the facilities of ViewSheet and the building of a simple model, five important commands are described and explained. All work only from the Command screen.

The completed model can be saved. If necessary, press ESCAPE to switch from the Sheet screen to Command mode. At the '=>' prompt, type the following:

```
SAVE  <model-name>
```

where <model-name> is a filename suitable for the current piece of work. Use a filename like 'GROWTH'. The whole spreadsheet is saved to the file on the current filing system.

Tape: The filename can be up to 10 characters long. After the SAVE command, 'RECORD then RETURN' is displayed on the screen. Put in a blank cassette and press the RECORD button, then press RETURN. Eventually, the Command prompt reappears, and the model has been saved.

Remember to label the cassette. Try to use the same name you saved it under to avoid confusion.

Disc: The model-name may have up to seven characters, plus drive name and directory name if necessary, for example ':1.M.GROWTH'. Make sure there is a suitably formatted disc in the drive. Keeping all ViewSheet files in a single directory is a good idea, perhaps M for models, as it helps to keep them separate from other types of file. Try not to use S for sheet, because ViewStore uses S for selection files.

ADFS/Net: Directory and file-names may be up to 10 characters long. It's best to keep all the ViewSheet files together in a directory called VSHEET, and use this as the current directory as follows:

```
*DIR  VSHEET
```

The filename could then be 'GROWTH'. Equally, it could be a full path-name '&.VSHEET.GROWTH'. Note that the directory 'VSHEET' must be created with *CDIR before it can be used:

```
*CDIR  VSHEET
```

When working with ViewSheet, save regularly. The saved copy is useful when the spreadsheet data is lost by accidentally replicating from a blank cell to a huge range of cells, or when the cat switches off the computer! The model can be given a name by typing:

```
NAME  <model-name>
```

The name is shown at the top of the Command screen, as 'Editing :1.M.GROWTH' for example. Once the model has a name then the SAVE command can be abbreviated to just:

```
SAVE
```

When a model has been saved, the sheet in memory can be cleared with the NEW command. ViewSheet makes all the cells blank and gets ready to build another completely new model.

Type:

```
NEW
```

Be careful, because unlike BASIC there is no OLD command that retrieves a sheet after it has been cleared. All trace of the old model is lost, so always SAVE first! After NEW, the Command screen shows 'Editing no file'.

The LOAD command can be used to reload an old spreadsheet back into memory from a file:

```
LOAD  <old-model-name>
```

<old-model-name> is the filename that was used to save the model. Take care, because loading an old sheet obliterates any model currently in memory. When reloaded, the model will be in exactly the state it was in when the SAVE command was used, and the model-name will be shown on the Command screen.

Tape: Ensure the cassette is rewound, and press the PLAY button. 'LOAD ""' loads the first file on the tape.

Disc: The filename may include the drive and directory names too, for example ':1.M.GROWTH'.

ADFS/Net: The filename could also include a directory name, or be a complete path-name, eg '&.VSHEET.GROWTH' or '$.VSHEET.GROWTH'. Consult the network manager if unsure of the best place to save a file on the network.

After loading, the Command screen is updated to show the new filename and the amount of memory left. It might show:

```
ViewSheet

Bytes Free 6512
Editing :1.M.GROWTH
Screen Mode 3
Printer default

=>_
```

and everything is ready for using the model again.

The last command needed to get going on ViewSheet is PRINT. To print out the sheet, first set up the printer. If you have a parallel interface printer (one that attaches with a wide ribbon-like cable), then you just plug it in.

Serial printers (plugged into the port marked RS423 on the back of the BBC micro), network printers (attached to a printer server elsewhere on the Econet) and Master Series need special attention as follows:

Serial: Select the serial printer by typing:

```
*FX  5,2
*FX  7,7
*FX  8,7
```

Serial printers work at different speeds. These commands give a speed of 9600 baud; if the printer needs anything different, then check the commands *FX7 and *FX8 in your micro's User Guide.

Network: Select the network printer by typing:

```
*FX  5,4
```

The *PS command may also be necessary to select the printer server, if it is not for example, station 235. If the network printer is not attached to station number 235, then consult the network manager.

Master Series: The Control Panel or the *CONFIGURE command can be used to set up the printer, so that the above commands are unnecessary. For example:

```
*CONFIGURE  BAUD  7
```

does the same as *FX 7,7 and *FX 8,7. *CONFIGURE PRINT 4 is the equivalent of *FX 5,4.

When the printer is set up, the sheet can be printed out by typing:

```
PRINT
```

It should be printed exactly as it looks on ViewSheet's Sheet screen. The model must be saved and in the micro's memory to print it out. Files can't be printed without loading them first.

One common problem that occurs is the print-out is all on one line. If this happens then enter:

```
*FX  6
```

and try again.

3 : Designing and Building a Model
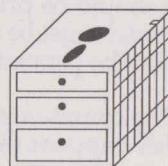
A Little Pre-planning

To get ViewSheet to help with a particular calculation, it is necessary to build a model that mimics the method used to make the calculation by hand, with paper, pencil and calculator. It is often not worth building the model if the calculation is to be performed only once. However, if the job has to be done regularly, a spreadsheet saves time. Building the model is similar to programming, but simpler because most of the difficult work has already been done. All you do is specify the data, the inter-relationships between the various pieces of data, and the formulae that link them together in various cells.

As with programming, planning pays. Before starting to type in a new model, it is useful to write down the types of data and all the formulae involved. Then sketch a preliminary layout for the sheet, showing where and in which cells you want to put the data, and take into account the rules below. The plan should resemble the way the same problem might be solved by hand on a big piece of paper. A typical sketch is shown in figure 3.1.

	1987	1986
Rent		
Rates		
Water Charge		
Maintenance		
	add up all costs for 1987	add up all costs for 1986

Figure 3.1. A sketch indicating a rough model layout.

After designing the rough outline, a start can be made on typing in the data, labels and formulae into the predetermined areas of the sheet.

Leave Room for Labels

There are two rules to follow when defining the layout for the data and formulae on the sheet. The first important rule is, never put the first piece of data in cell A1; always leave a few rows and columns around the edges of the sheet so that any labels can be inserted. In the trial model 'GROWTH', the first value went into cell C3, so two rows and two columns were left free. This also applies within the body of the model: leave plenty of room for labels near cells containing formulae.

Direction of Recalculation

The second layout rule is that when entering your figures, you should generally proceed downwards and right, across the sheet. In particular, a formula should only make references to cells that are either in a lower-numbered row (that is, higher up the sheet) or the same row but an earlier column. As an example, a formula in cell C7 may refer back to any of the cells in rows one to six, plus A7 and B7, but should not refer forward to D7, E7, or any succeeding cells. Figure 3.2 illustrates this.

yes	yes	yes	yes	yes
yes	yes	FORMULA	no	no
no	no	no	no	no

Figure 3.2. Cell values a formula may refer to.

This is necessary because of the way ViewSheet recalculates the values of all the cells whenever a change is made to the sheet. The first to be calculated is the altered cell, then A1, then B1, continuing through all of row one to IU1 before calculating the value of A2, B2 and so on. The direction of recalculation is from left to right along each row in turn, from row one to row 255.

Skip to the bottom of the next page if the next bit looks too technical. This direction of recalculation might not seem important. After all, every time a cell is changed, that particular cell is evaluated immediately. Then the sheet is recalculated, so an up–to–date value for the changed cell is used throughout. However, when a cell is altered, formulae refering to that cell are not recalculated immediately, but

must await their normal turn. Therefore, cells making a forward reference to these other formulae will be evaluated incorrectly.

To demonstrate the importance of this, load the trial model 'GROWTH', and enter these new cells:

> cell E1 CHANGE RETURN SHIFT–f8
> cell E4 (C8 / D8 - 1) * 100

This makes E4 show the correct percentage growth of the average figure between last year and this, 20.4545. Now alter cell D3:

> cell D3 80

The total and average figures for last year are updated correctly, to 159 and 53, so last year and this now show no overall change. But Cell E4 still shows 20 per cent growth! When this cell was evaluated, the old average was still in cell D8.

To be evaluated correctly, the percentage change formula should be placed below the average figures to which it refers. Delete E1 and E4 with SHIFT–f9, place a new formula in cell C10 and label in B10:

> cell B10 CHANGE
> cell C10 (C8 / D8 - 1) * 100

Now, even when the data is altered, the change figure will always be correct; alter D3 to 74 to check, and the sheet should look like figure 3.3, indicating about four per cent growth from year to year.

```
VA   SLOT=C10
CONTENTS=(C8/D8-1)*100

0      ......A......B......C......D......E
......1              YEAR     This   Last
......2
......3              APRIL     68     74
......4              MAY       46     41
......5              JUNE      45     38
......6
......7              TOTAL    159    153
......8              AVERAGE   53     51
......9...........................................
.....10              CHANGE  >3.92157
```

Figure 3.3. A correct place for the CHANGE formula.

In fact, formulae often do refer to cells they should not. The best advice is that it is fine to refer forwards to ordinary values, but never forwards to a cell that contains another formula.

Another problem occurs when a cell formula refers to itself. This can happen very easily when a column total is calculated. Imagine cell C7 containing the formula C2 C7, instead of C2 C6. It's harder to spot when the self-reference is indirect: when one cell refers to another and the second refers to the first. This is a *circular reference*, but it can't happen unless one of the cells first breaks the rules and makes a forward reference.

Displaying Numbers on Screen

The way in which numbers are displayed in ViewSheet is called the *number format*. For example, the number 37.5 can be displayed as 37.5, or 37.5000, 3.75E1, or even 38. Each is a different format.

Formats for Individual Cells

The value of the formula in cell C10 of figure 3.3 is shown as 3.92157. The precision with which the value is displayed can be altered using the EDIT SLOT FORMAT function. Make C10 the active cell and press f6; the current *cell format* appears on the Edit line where it can be altered:

```
f6
Format?
FRM
```

The format FRM has three elements. The first indicates whether the value displayed should have a floating decimal point, or a fixed number of decimal places: F, D1, D2 and so on. The second, L or R, shows if the value should be placed to the left or right of the cell. The third controls whether negative numbers should be shown with a minus sign, M, or in brackets, B: that is -12.34 or (12.34). Brackets are used widely in financial work.

25

Try a few of the following examples out in cell C10:

D1RM	C10 shows	3.9
D2RM		3.92
D3LM		3.921
FRB		3.9215
D0RM		4
D6RM		%

In the last of these examples, a cell showing '%' indicates that the value associated with the cell can't be displayed in the chosen format. The cell is only seven characters wide. The format asks for six decimal places, which requires eight characters (3.921569), so the slot is too narrow. If a value doesn't fit the format, then it is shown in exponential form. If that doesn't fit either, then the per cent sign is shown.

Change the format back to D0RM. The D0 indicates rounding to the nearest whole number - 1.5 is rounded upwards to 2, 1.49 is rounded down to 1 - and so the value of the C10 formula is rounded to 4. Briefly return to the Command screen and save the whole model in this state.

However precisely a value is displayed, ViewSheet stores it internally to full nine figure accuracy. This can be demonstrated by putting the formula D0 * 100 temporarily in some other cell. The value of this is not 400, but 392.157. Cell C10 shows the rounded value, but the true value is used in other calculations.

Changing the Format of all the Cells

Cell formats may be changed individually, but alterations can also be made to the whole screen, by using the EDIT WINDOW function. Press f1, ViewSheet then asks for the *window number*. This is the number that appears in the upper left corner of the sheet area of the screen, below the status area, and at present this should be 0. Finally, the current *window definition* appears on the Edit line:

```
f1
Window?
0
Wi  TopL   BotR   Pos Cw  Bw Fmt   Opt
0   A1     I19        7   7  FRM
```

This definition line can be edited in the normal way by using CTRL–RIGHT and CTRL–LEFT to move the flashing cursor along the line, and overtyping items as required. Wi is the window number, and should not be changed for now. The TopL and BotR cell references show the part of the sheet that is displayed in the window, and again they shouldn't be altered.

Cw is the width of the columns. By default, each cell can display seven characters, but this can be altered. If Cw is edited to 10, then wider cells will be displayed, but there would be space for fewer columns across the screen. ViewSheet automatically decreases BotR to take this into account. Cw can vary between three and the total width of the screen: narrow columns squeeze more cells into the display, wider ones allow longer labels or more precise number formats. If the per cent sign is shown, then increasing the column width may allow the full value of the number to be displayed.

Bw is the width of the dotted border down the left side of the screen. This can vary between two and 15.

Fmt is the *window format*, and it follows the same rules as the format for individual cells. Load the 'GROWTH' model, and try changing the window format to D1RM:

```
Wi TopL  BotR  Pos Cw  Bw Fmt   Opt
 0  A1    I19       7   7  D1RM
```

After pressing RETURN, the cells change so that all the figures have a single decimal place. The exception to this should be the C10 cell containing the CHANGE formula. This should still have an individual slot format D0RM set - these slot formats take precedence over the window formats. To delete the slot format for C10, begin to edit it (with f6) and delete the whole thing (by pressing f3 and then RETURN). This makes C10 follow the format rules set for the window as a whole.

Doing Arithmetic

Simple arithmetic can be done on the sheet. Ordinary numbers, or constants, are linked together with *operators*.

27

The standard operators, in order of decreasing *precedence*, are:

^	raise to the power of
*	multiply
/	divide
+	add
−	subtract

Precedence means that the multiplication in a formula is normally done before any addition - it has higher priority. If you need to put operators in a different order, brackets may be used. For example, 10 * 3 + 5 is 35, but 10 * (3 + 5) is 80.

Formulae can also contain conditions which may be 'true' or 'false': 10 > 3 is true, whereas 10 = 3 is false. True has the numerical value of one, and false has the value zero. The usable conditional operators are as follows:

=	equal to
< >	not equal to
>	greater than
> =	greater or equal to
<	less than
< =	less or equal to

These always have a lower priority than the mathematical operators, so 10 * 3 > 10 + 3 is evaluated as ((10 * 3) > (10 + 3)). The value of this formula is 1; 30 is greater than 13, so the formula is true.

Built-in Functions

All the usual arithmetic, trigonometric and logarithmic functions are available using ViewSheet. Many will be familiar from calculators and from BASIC, and, with very few exceptions, they work exactly as they do in BASIC. An example of a formula using one of the built-in functions is:

```
SQR (D5 - D4)
```

This formula gives the square root of the difference between the values in cells D5 and D4. Note that D5 must be greater than D4, otherwise an error occurs as ViewSheet can't find the square root of a negative number.

Note, all the ViewSheet functions need brackets around the *argument* (the argument is D5 – D4 in the example). If they are omitted, the formula is treated as a label. Spaces between the function and the opening bracket are optional.

Special Spreadsheet Functions

All spreadsheets have an extra group of functions especially helpful in constructing models. The way they work varies between spreadsheets, and this often forms the major difference between rival packages.

ROW and COL

The simplest among ViewSheet's special functions are ROW and COL. The value of each of these depends upon which cell they are in. A ROW function in cell B7 has the value seven because B7 is in the seventh row; in cell D10, it has the value 10.

In cell B7, COL does not have the value B, but two, because B is the name of the second column. Clearly, in cell D10, COL gives the value four. In cell AA10, the value of COL is 27, because AA comes after column Z (which is column 26). The last column IU has a COL value of 255.

Figure 3.4 illustrates a 12 by 12 multiplication table constructed by putting the formula 'ROW * COL' into 144 cells on the spreadsheet. As shown, the value of this formula in cell K3 is 11 * 3 (K is the 11th column), or 33.

```
VA   SLOT=K3
CONTENTS=ROW*COL

0        ...A....B....C....D....E....F....G....H....I....J....K....L
......1     1    2    3    4    5    6    7    8    9   10   11   12
......2     2    4    6    8   10   12   14   16   18   20   22   24
......3     3    6    9   12   15   18   21   24   27   30 > 33   36
......4     4    8   12   16   20   24   28   32   36   40   44   48
......5     5   10   15   20   25   30   35   40   45   50   55   60
......6     6   12   18   24   30   36   42   48   54   60   66   72
......7     7   14   21   28   35   42   49   56   63   70   77   84
......8     8   16   24   32   40   48   56   64   72   80   88   96
......9     9   18   27   36   45   54   63   72   81   90   99  108
.....10    10   20   30   40   50   60   70   80   90  100  110  120
.....11    11   22   33   44   55   66   77   88   99  110  121  132
.....12    12   24   36   48   60   72   84   96  108  120  132  144
```

Figure 3.4. 12 x 12 table constructed with ROW and COL.

Summing up a Range

Another straightforward group of functions are related to ranges (parts of a single row or of a single column). The simplest of these is the SUM function, which doesn't really exist! However, there is a shorthand form of SUM. To add up the values in the range between D5 and D10, the formula is just:

```
D5 D10
```

which means:

```
D5 + D6 + D7 + D8 + D9 + D10
```

Many other spreadsheets have an explict SUM function, but this is unnecessary in ViewSheet.

One point to consider is whether the range specified for summing includes any blank cells or any labels. The values of these cells are taken to be zero: they don't count towards the sum.

Related to the SUM function is AVERAGE (n1, n2, …). This function takes as its argument (the bit in brackets) a *list*. The list can contain any combination of constants, cell references, other functions or ranges. Items in the list are separated by commas. For example:

```
AVERAGE (125, C2 C5, D2 D5, LOG (E2))
```

would add together 125, the values of the four cells in each of the two ranges C2 to C5 and D2 to D5, and the logarithm of the value in cell E2, and finally divide by 10, the total number of items in the list.

The range C2 C5 contains four items, but what happens if one of the four cells in the range is blank? As with SUM, a blank cell makes no contribution to the adding up, but it does still get counted as an item. So the AVERAGE of the range is not the average of the cells with values in them. AVERAGE underestimates if there are any blank cells or labels in the range, so you must make sure that these are not accidentally included in ranges.

The MAX (n1, n2, …) and MIN (n1, n2, …) functions work in just the same way as AVERAGE. They work out the single largest or smallest value in

their list. Once again, empty cells or labels must be avoided, especially with MIN, as they have a notional value of zero.

There are five more special functions, CHOOSE, IF, LOOKUP, READ and WRITE, which will be described in later chapters.

Naming Rows and Columns

Sometimes, using cell references in formulae can look a bit cryptic - just what does F12 F16 mean? There is a way of making this a little easier to interpret, by using the COLUMN HEADING and ROW HEADING function keys.

From the Command screen, reload the 'GROWTH' model and switch to Sheet mode. Make cell C3 active by moving the cell cursor, and press SHIFT–f3. A 'Column heading?' prompt will be displayed, so type in a heading for the whole of column C. The column contains the figures for this year, so an appropriate heading is 'This', type:

```
SHIFT–f3
Column heading?
This
```

Now do this for the whole of row three too:

```
SHIFT–f4
Row heading?
April
```

The status area of the screen now shows the following:

```
VA   SLOT="This"APRIL"
CONTENTS=68
```

The slot is no longer called C3, but is known by its column and row headings: "This"April".

Other headings can be put on the sheet in a similar way, and eventually the sheet should look like figure 3.5. Spaces can be included in the headings, for example, row 10 is headed YEAR TO YEAR.

The screen shown also has the border width (Bw in the window definition) changed to 13, so that there is room for the longest row headings.

```
VA  SLOT="This"April"
CONTENTS=68
```

```
Wi TopL BotR Pos Cw  Bw Fmt  Opt
0  A1   H19      7  13 FRM
0             .......A.......B....This....Last.......E
........YEAR           YEAR     This    Last
.........2
.......April           April >    68      74
........May            May        46      41
.......June            June       45      38
.........6
.......TOTAL           TOTAL      159     153
.....AVERAGE           AVERAGE    53      51
.........9
.YEAR TO YEAR          CHANGE     4
```

Figure 3.5. Final trial sheet.

Once the rows and columns are headed like this, then a cell can be
referred to in a formula by any combination of its row number, column
letter and headings, or by pointing. For example "This"April", C"April"
or "This"3 all refer to cell C3, but the first gives most clue to what the
figure really represents. Formulae are improved too:

```
CONTENTS="This"April""This"June"/3
```

is a much more comprehensible description of the contents of C8 than:

```
CONTENTS=C3C5/3
```

When using headings in formula or with the GO TO SLOT function key,
double quotes (SHIFT–2) should be used to separate the row and column
headings from each other and from other symbols. In this example,
'"This"April"This"June" /3' is the minimum ViewSheet could recognise.

Headings can be temporarily switched off. Press ESCAPE to go to
Command mode, then type the command:

HEADINGS OFF

Back in Sheet mode, the row and column headings are not shown. They
can be restored with the command:

HEADINGS ON

Taking Protective Measures

When set up, the formulae in a model should not be changed
inadvertently. Imagine the havoc if the builder's quotation spreadsheet

mentioned earlier was altered to contain the wrong formula for the number of bricks!

Individual cells can't be protected from alteration, but whole rows or whole columns can be locked to prevent any of their cells being altered accidentally. To lock a row, move the cell cursor to any cell on that row and press SHIFT–f6. This is the PROTECT ROW function, and it stops any cell in that row being altered.

On our sample 'GROWTH' spreadsheet, it is wise to protect rows seven, eight and 10 in this way. When these rows are locked, the dots in the border containing the row headings change to underlines, indicating the specially protected rows. Any attempt to alter the cell contents in a locked row results in the message 'Protected' appearing in the status area. Pressing SHIFT–f5 does the same for all the cells in a single column. When a cell is protected, it cannot be changed by typing in new contents, by editing its contents or by replication. An attempt to replicate to a protected cell fails, though the rest of the replication may succeed. In this case, no error message is shown, which can cause confusion.

If it's vital to change a protected cell, it can be done. A second press of SHIFT–f5 or SHIFT–f6 unlocks an individual column or row, and changes the border back to dots. You can also temporarily unlock all the cells on the sheet. To do this, press ESCAPE to switch to Command mode, and type in the following:

```
PROTECT OFF
```

Now on returning to Sheet mode, any cell can be modified. Entering PROTECT ON in the same way restores the previous protection. If PROTECT on its own is entered, then ViewSheet shows whether protection is currently on or off.

Replication Round-up

In the worked examples so far, replication has been used to copy from a single cell to a single cell and from a cell to a range of cells, using both absolute and relative replication.

In fact, replication also works from a range to a single cell and from one range to another range. Figure 3.6 summarises the seven available options; note that each replication involving a column range has a

similar row-wise variation. The options missing from the set in figure
3.6 are the impossible replications from a row range to another row
range, and from a column range to another column range. When
replicating from one slot to another, any cell references in the slot
copied must be specified as absolute or relative; the prompt is:

```
R)elative, N)o change?
```

This may have to be repeated for each cell reference in the source slot.
To replicate the formula D2*F2–G2, either R or N must be pressed three
times because there are three cells.

When copying from a range, then this sequence is followed for each of
the slots in the range. That is, the replication from D3 D5 – E3 is treated
as three separate cell-to-cell replications, D3 to E3, D4 to E4 and so on.
The 'R)elative, N)o change?' question must be answered separately
for each of the three. Curiously, the questions are presented in reverse
order; the formula in D5 is copied first, and that in D3 last. If this seems
confusing, remember that the status area always shows which slot is
currently being copied, and the full contents of that slot, in the top two
lines. This can be tried with the 'GROWTH' model; delete the two cells
D7 and D8 by pressing SHIFT–f9. Now replicate the range C7 C8 to D7:

```
f0
From - To?
C7 C8 - D7
```

At this point, the status area shows:

```
VA  SLOT=C8
CONTENTS=C3C5/3
R)elative, N)o change?
C3C5/3
```

This shows that the bottom of the C7 C8 range (cell C8) is copied first.
Completing this replication involves pressing R five times, once for each
cell reference in each of the formulae.

slot to slot	eg. B17 – E17	

slot to range	eg. B17 – E17E19	
	— or —	
	B17 – E17G17	

(three cells copied)

range to slot	eg. B17B19 – E17	
	— or —	
	B17D17 – E17	

(three cells copied)

range to range	eg. B17B19 – E17G17	
	— or —	
	B17D17 – E17E19	

(nine cells copied)

formula
refers here

relative replication

formula
refers here

absolute (no change) replication
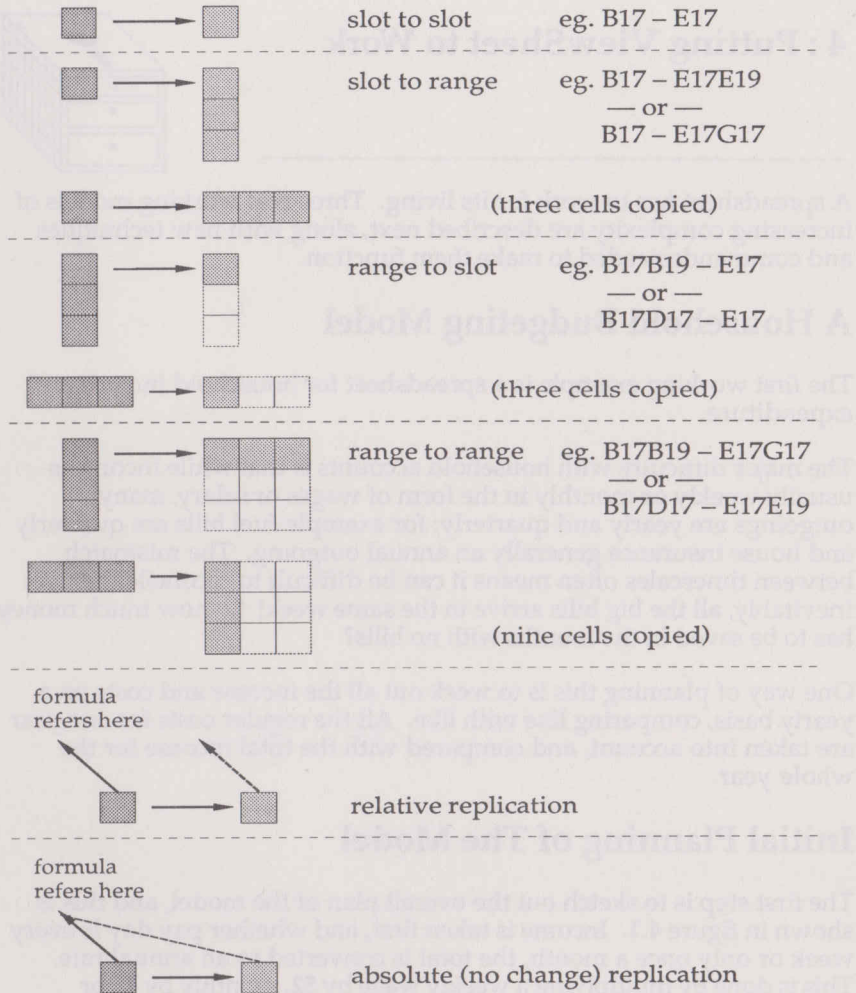
Figure 3.6 Summary of replication options

35

4 : Putting ViewSheet to Work
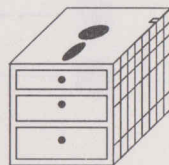
A spreadsheet has to work for its living. Three real working models of increasing complexity are described next, along with new techniques and commands needed to make them function.

A Household Budgeting Model

The first working example is a spreadsheet for household income and expenditure.

The major difficulty with household accounts is that while income is usually weekly or monthly in the form of wages or salary, many outgoings are yearly and quarterly; for example fuel bills are quarterly and house insurance generally an annual outgoing. The mismatch between timescales often means it can be difficult to reconcile the two. Inevitably, all the big bills arrive in the same week! So how much money has to be saved in the months with no bills?

One way of planning this is to work out all the income and costs on a yearly basis, comparing like with like. All the regular costs for one year are taken into account, and compared with the total income for the whole year.

Initial Planning of The Model

The first step is to sketch out the overall plan of the model, and this is shown in figure 4.1. Income is taken first, and whether pay day is every week or only once a month, the total is converted to an annual rate. This is done by multiplying a weekly wage by 52, monthly by 12 or quarterly by four. Note that the income figures that should be used are net, or take-home pay after deduction of tax and national insurance.

The expenditure columns are taken in the same way. Finally the two total figures are compared.

36

INCOME	WEEKLY MONTHLY QUARTERLY YEARLY	ANNUAL EQUIVALENT
JANET		
JOHN		↓
		GRAND TOTAL INCOME

EXPENSES	WEEKLY MONTHLY QUARTERLY YEARLY	ANNUAL EQUIVALENT
BILL A		
BILL B		
BILL C		↓
		GRAND TOTAL INCOME

Figure 4.1. Initial sketch for household budget model.

Think about number formats. The normal format is FRM (which means floating decimal point, right-hand side of the cell, minus signs if negative), but this doesn't suit this model. When displaying money values two decimal places are often required, so the D2RB or D2RM functions would be better.

The next thing is to note down all the household's sources of income, and all its regular costs. Here is a quick list to start off with:

INCOME	wages	salary	interest	share dividends
COSTS	rent	rates	mortgage	home insurance
	water	gas	telephone	electricity
	petrol	car tax	loans	car insurance
	food	fares	laundry	life insurance

Of course, not all of these will apply, and you may have different outgoings - if you have three cats, vet's bills and feeding them have to be taken into account!

Building the Model

The basic aim of the model is to sum the total income and the total expenditure. Therefore the first labels to put in denote columns for weekly, monthly, quarterly or annual income. These should be put in cells A3 to F3, as shown in figure 4.2. Note that some of the labels are at the left end of their cells, some at the right; JUSTIFY LABEL (key f8) can be used to swap them across.

```
LA   SLOT=F3
CONTENTS=Annual


0        ......A......B......C......D......E......F
......1                   HOUSE BUDGET
......2
......3 INCOME    Week   Month Quarter   Year> Annual
```

Figure 4.2. Beginning a household budget model.

Now a row should be set aside for each source of income. The first in this example is Janet's salary, and this is labelled in cell A5. Janet is paid £520 net monthly, so her income is put in cell C5, the monthly column. John takes home £131 per week, so this is put in cell B6, the weekly column.

Column F is to hold the annual equivalent of each income source; if the income is weekly then multiply it by 52 to give the annual income, if it is quarterly then multiply it by four. The formula in cell F5 can be entered as follows:

```
52 * B5 + 12 * C5 + 4 * D5 + E5
```

Remember to point at B5 and the other cells whilst doing this; use the SHIFT–COPY facility instead of typing 'B5'. As B5, D5 and E5 are blank their value is zero, so the total is merely 12 * C5, or £6240. This is the annual equivalent of £520 per month.

This formula can be replicated from one cell to another, that is from F5 to F6. Press f0 and reply:

```
F5 - F6
```

to the prompt. Then press R as needed. This is relative replication; each formula should refer to the cells to its left, not just the same cells as the original formula. The income section is completed with some more

labels and, in cell F8, the formula F5 F6 to give the sum of the cells between F5 and F6. Figure 4.3 shows the model at this stage.

```
VA   SLOT=F6
CONTENTS=52*B6+12*C6+4*D6+E6
0       .......A.......B.......C.......D.......E.......F
.....1                      HOUSE BUDGET
.....2
.....3 INCOME    Week   Month Quarter    Year  Annual
.....4 ------
.....5 Janet            520                     6240
.....6 John     131                        >   6812
.....7                                          ------
.....8                             TOTAL INCOME 13052
.....9
```

Figure 4.3. The income half of the budget model.

The expenditure half of the model can be constructed in a similar fashion. First, type in any labels necessary using the model sketch as a guide. There should be an EXPENSES label in A10, and below this a list of expenditure headings. For now, type in only five spending categories: mortgage (or rent), rates, gas, telephone and electricity.

Now the formulae for the annual equivalents can be put into column F. These formulae are just the same as those in the income section. Again, relative replication should be used, but this time from a single cell to a group or range of cells. Press REPLICATE (key f0), and type:

    F5 - F12 F16

Then press R to indicate relative replication.

The final formula in this section should be in row 18, and it denotes the sum of the five expenses rows:

    F12 F16

This formula is, of course, the equivalent of F12 + F13 + F14 + F15 + F16.

Now the actual costs of the five items can be typed into the relevant cells. A mortgage is paid monthly, so the cost goes in cell C12; gas bills are quarterly, so the average size of bill should go in D14.

At this stage it is convenient to alter the number display format. This is done by editing the window definition. As this model shows amounts of money, it is an advantage to show all figures with exactly two decimal

places. The format needed is D2RM. To alter the window definition press f1 (EDIT WINDOW), and answer the 'Window?' prompt by pressing 0; then edit the definition:

```
Wi  TopL    BotR    Pos  Cw   Bw  Fmt      Opt
0   A1      I26          10   7   D2RM
```

At the same time as altering the number format, the column width can be increased so that labels with up to 10 letters can be shown in full. Remember this will leave room for fewer columns across the screen. After this section is completed, the sheet should look like figure 4.4. Remember to save the model, perhaps using the filename 'HOUSE':

```
SAVE HOUSE

VA  SLOT=F18
CONTENTS=F12F16

0       ..........A..........B..........C..........D..........E..........F
......1                               HOUSE BUDGET
......2
......3  INCOME          Week     Month   Quarter      Year      Annual
......4  ------
......5  Janet                    520.00                         6240.00
......6  John            131.00                                  6812.00
......7                                                       ------------
......8                                          TOTAL INCOME   13052.00
......9
.....10  EXPENSES
.....11  --------
.....12  Mortgage                 371.66                         4459.92
.....13  Rates                             235.47                 941.88
.....14  Gas                                52.98                 211.92
.....15  Phone                              41.56                 166.24
.....16  Electricity                        37.81                 151.24
.....17                                                       ------------
.....18                                       TOTAL EXPENSES  >  5931.20
.....19                                                       ------------
.....20                                          NET INCOME    7120.80
.....21                                                       ============
.....22                                   NET WEEKLY INCOME     136.94
.....23                                                       ============
```

Figure 4.4. The completed household budget model.

Inserting Columns and Rows

In figure 4.4, only five major expenses rows are shown, but other categories from the preliminary list can be inserted easily into the model. First, make F18 the active cell, and note that the status area says 'CONTENTS=F12F16'. This cell totals up all the individual expenses.

40

Move the white cell cursor to F14, and press INSERT ROW (function key SHIFT–f2). A new blank row 14 is inserted above the old one, and the old rows 14 to 21 are shuffled down. Gas costs or any of your other outgoings can be put in row 15, and the new row 14 could be used to record the annual water rates bill, for example. Add as many rows as needed to get an accurate picture.

The important point to notice is that the formula in F19 (the old F18) now says 'CONTENTS=F12F17'. ViewSheet tries to amend the formula to take into account the extra row, so the sum now includes the new value. The new water rates row was inserted into the middle of the range F12 F17. Now check what happens if a row is inserted at the beginning or the end of the range. Any new rows above row 12 (Mortgage) or below row 17 (Electricity) are ignored because they are outside the range in the original formula.

New blank columns can be inserted in exactly the same way as rows, using INSERT COLUMN (key SHIFT–f1). The new row or column always appears at the cell cursor position.

Deleting Rows and Columns

Blank rows and columns are usually a good thing. They serve to divide up the sheet into logical blocks; row nine in figure 4.4 separates the income area from the expenditure. However, there may be too many blank areas. Rows and columns can be removed, along with the cells they contain, by pressing CTRL–f2 or CTRL–f1 while the cell cursor is in the correct column or row. These keys are marked DELETE COLUMN and DELETE ROW.

Before deleting anything, check the area is really blank. Run the cell cursor along the row; remember not all of the row is shown on screen at once, and there may be valuable data in the invisible section. So always be sure and check off-screen too.

While deleting a row or column, the model must be temporarily unlocked by using PROTECT OFF in Command mode mentioned earlier. When protection is removed, switch to Sheet mode, position the cell cursor and press DELETE COLUMN or DELETE ROW. The area is deleted and adjacent cells are shuffled up to fill the gap. Remember to relock the sheet with the PROTECT ON command.

As with INSERT ROW and INSERT COLUMN, ViewSheet attempts to adjust the cell references in all formulae to compensate for the changes, but fails if the deleted row or column is referred to directly. For example, if a formula contained a reference to cell C5, then deleting row five would cause problems: the reference would remain C5, but C5 would then refer to the old C6! This should not cause problems providing that formulae never refer to blank cells, and nothing except blank cells are ever deleted.

Extending the Model

On the budget model, insert any other types of regular outgoings necessary. The last two formulae, shown in cells F20 and F22 in figure 4.4, are the Net Income (Total Income - Total Expenses) and the Net Weekly Income (Net Income / 52). This final figure shows how much 'spare' money there is in Janet and John's pockets each week - money not already earmarked for something. After inserting any new items, save the model again.

The model can be used to check that any new commitment that Janet and John make, perhaps to a new savings plan or a bigger car, will not take away too much of their income. The figures can be varied at will and the overall effect of the change can be seen at once.

Statistics with ViewSheet

Descriptive statistics are numbers that summarise a large group of measurements. Best known is the average (or arithmetic *mean*), the sum or total of the measurements divided by the number of measurements. ViewSheet has a special AVERAGE function to find the mean of a group of figures.

Another useful description of a set of numerical data is the *standard deviation*. This illustrates the amount of *variation* or *dispersion*, ie, how much they differ from their average. This is also sometimes quoted as the *coefficient of variation*, which is the standard deviation as a percentage of the mean. Wildly varying things like the heights of a class of schoolchildren have a high coefficient of variation, whereas steady figures like the weights of 30 wooden building blocks of the same size have a low coefficient.

The Pebble Project

This section describes the use of ViewSheet to work out the average and standard deviation of a set of figures. The example chosen is an investigation of the variation of pebble size on a beach; first the range of sizes at one place and second the range of average sizes up and down the beach. At spring low tide, the largest area of beach is exposed, from the finest sand through coarse sand, fine gravel and pebbles. At each part of the beach, a sample can be taken: a little bag of coarse sand, a big bag of pebbles, a bucketful of shingle and so on. The individual sizes of 30 to 40 pieces from each sample can be measured. Even sand grain lengths could be estimated, using a hand lens to look at grains spread out on millimetre graph paper. Gravel, shingle and pebbles should present no problems.

Now for the spreadsheet. To work out a standard deviation for one sample, a worksheet is often laid out like figure 4.5. First come the measurements themselves, then a column showing the difference between each figure and the overall average. Next a column showing the differences squared. The mean of these squared differences is found; this is known as the *variance*. The standard deviation is the square root of the variance. In fact, there is a quicker way requiring only two columns, but it's much harder to understand!

Measurements	Deviation from mean	(Deviation from mean)^2
1	−4	16
2	−3	9
3	−2	4
4	−1	1
5	0	0
6	1	1
7	2	4
8	3	9
9	4	16

Total = 45   Total = 60

Number of measurements = 9   so variance = 60÷9 = 6.7
So mean = 45÷9 = 5   ∴ Standard Deviation = $\sqrt{6.7}$ = 2.6

Figure 4.5. Rough layout for a statistical model.

This structure can be duplicated exactly on the spreadsheet. Into column B goes the list of measurements. The next column should be set aside for the differences, then the third column for the squares.

Using Auto-Entry to Build the Pebble Model

Figure 4.6 shows a few shingle length measurements, together with the first two formulae. Although the example sheet shows only nine, there should be at least 30 measurements for the standard deviation to be useful. When typing in the measurements, AUTO ENTRY might help. This makes the cell cursor move each time RETURN is pressed, so it's ready for the next number. Pressing CTRL–f0 makes an 'R' flag appear in the top left corner of the screen; the cursor moves right after RETURN is pressed. A second press of CTRL–f0 makes a 'D' flag appear, and the cursor moves down. A third press switches AUTO ENTRY off again if necessary. In this case, where there is a column of figures to type in, D is most useful.

```
            VAD  SLOT=D6
            CONTENTS=B6-AVERAGE (B6B14)


        0   ........A.......B........C.......D.......E
        ......1              BEACH  PEBBLE  PROJECT
        ......2              -----  ------  -------
        ......3              length         diff
        ......4              cms            cms
        ......5
        ......6              11.1    >0.94444
        ......7              12.3
        ......8              7.9
        ......9              10.4
        .....10              10.8
        .....11              9.6
        .....12              10.2
        .....13              10.1
        .....14              9
        .....15              --------
        .....16              10.1556
```

Figure 4.6. Starting the Beach Pebble Project.

Cell B16 contains the formula AVERAGE (B6 B14), the mean of the measurements. Cell D6 should contain the difference between the first measurement (in B6) and the mean. But referring to the mean in B16 would be a forward reference to another cell containing a formula! Repeating the AVERAGE function in cell D6 (so the status area says CONTENTS=B6-AVERAGE (B6B14)) avoids any problem. Remember that referring forward to an ordinary value cell is not against the rules, so with cells B6 to B14 it won't cause any problems.

Now the whole column of difference formulae can be written in by replication. This is a mixture of relative and absolute replication. Cell D6 contains the formula B6 - AVERAGE (B6 B14); the B6 part refers to the cell to the left, relatively, but the AVERAGE section should always refer to the range B6 B14 absolutely, no matter where in the spreadsheet the formula is replicated to:

```
f0
From - To?
D6 - D7 D14
R)elative, N)o change
B6-AVERAGE(B6B14)
R N N
```

R is pressed when the first B6 is highlighted on the edit line, and N when the second B6 and B14 are highlighted. That completes the second column in the model.

The third column can be constructed in a similar way: first type in the top formula, then replicate to the rest of the column. The column should contain the squares of the differences in column D, so the formula in cell E6 is D6 ^ 2. Note that while some of the differences are negative, their squares are all positive. Figure 4.7 shows all three columns complete. Note also the window definition has been edited to change the number format to D1RM and widen the columns to nine characters, making the display neater.

```
VA  SLOT=E18
CONTENTS=E17/B16*100
```

0	........A........	.B.......	...C........	.D........E
......1		BEACH	PEBBLE	PROJECT
......2		------	------	-------
......3	Shingle	length		diff	diff^2
......4	-------	cm		cm	cm^2
......5					
......6		11.1		0.9	0.9
......7		12.3		2.1	4.6
......8		7.9		-2.3	5.1
......9		10.4		0.2	0.1
......10		10.8		0.6	0.4
......11		9.6		-0.6	0.3
......12		10.2		0.0	0.0
......13		10.1		-0.1	0.0
......14		9.0		-1.2	1.3
......15					-------
......16	mean	10.2		variance	1.4
......17		standard	deviation	1.2
......18		coeff. of	variance▶	11.7

Figure 4.7. The three columns complete.

45

The final touches can be added to this model, by adding the three formulae in cells E16 to E18. The variance formula in E16 is the sum of the squared differences, E6 E14. The standard deviation in E17 is the square root of the variance, SQR (E16). The coefficient of variation is the standard deviation expressed as a percentage of the mean, so the formula in E18 is E17 / B16 * 100. In the example, the coefficient is 11.7%. At this point in the exercise save the model, perhaps using the file-name 'BEACH'.

Recalculation Mode

If the model gets really large, then each time a new value is put in, three pulsing dots may be seen in the upper left corner of the screen. This shows that ViewSheet is working, recalculating the whole model. With a really big sheet, this may take up to five seconds.

Normally ViewSheet recalculates the entire sheet automatically every time a cell is changed. This can make the process of entering data tedious and unnecessarily slow, because of the wait while a large model is being reworked. Automatic recalculation can be switched off by pressing RECALCULATE MODE (key SHIFT–f0). The flag in the top left corner changes from 'A' for automatic to 'M' for manual, and now data can be entered into a slot without any noticeable working delay.

This of course means that the model will not necessarily display the latest up-to-date cell contents, because if a cell is changed, then the effects will not ripple across the model. They only do this when the model is recalculated. The whole model can be recalculated at any time, by pressing SHIFT–f7, RECALCULATE. This brings all cells up to date.

Automatic recalculation can be restored by a second press of SHIFT–f0.

Extending the Pebble Model

If the gravel and sand samples from other parts of the beach are also required on the model, then all the formulae and labels can be copied across (relatively) to other parts of the spreadsheet, say to columns AA to AE. Then just type in the new measurements and the work is done! Room for extra measurements can be created by inserting extra rows. But remember, the space will be inserted into columns A to E too! For this reason, if there are different numbers of items in each sample, then

replication of the labels and formulae should be downwards, towards A100 and E100 instead of across to columns AA and AE.

If a new area has been copied, and there is too much room, then delete rows so that the blanks do not interfere with the AVERAGE function.

Another way to go about this is to use a separate model for each sample; load the 'BEACH' sheet, and delete the data in cells B6 to B14. Replication of a blank cell is the quickest way of doing this; cell A1 is blank, so:

```
f0
From - To?
A1 - B6 B14
```

Now the model contains no data, but all the formulae are still there, ready and waiting. Now save the model using a different name, perhaps 'PEBmask'. This model without data can be loaded in each time a standard deviation is needed; only the new data need be typed in and the name changed again. This type of sheet without data but containing fiormulae is often called a *mask*.

Doing Business with Viewsheet

A spreadsheet is the ideal tool to help with laying out columns of figures. Perhaps the commonest example is when a budget has to be prepared. Several sub-costs have to be totalled on a product or functional basis. All the revenue has to be estimated from projected sales and prices.

The process of setting out such a model is akin to the Pebble Project. A typical layout sketch is shown in figure 4.8, which shows a plan for a group of similar products, say a range of books. The areas where data has to be entered into the model are shown, together with descriptions of the necessary formulae. Most of the formulae simply multiply or sum other cells. For example, the expected revenue from one particular product is the unit price of the product multiplied by the number of units that should be sold. The total expected revenue is the sum of the expected revenue for each product.

A more complex formula is that for the unit selling cost of a product - the cost of selling (not manufacturing) one book. This is made up of the cost of promotion of the book title, plus the cost of distribution for that book. In the model, the distribution costs are not known for individual

titles; only the total spending on distribution is used. Therefore each title bears a proportion of the total cost of distribution, according to the proportion of total book sales it accounts for. A typical formula for a title in row 10 might be A4 * (C10/(C9 C11)) + E10.

Total Distribution Costs

☐

	Unit Price	Unit Sales	Unit Costs	Promotion Costs	Total Selling Costs	Gross Revenue	Gross Costs	Net Revenue
Book 1					Distribution x(unit sales+ Total Sales) +Promotion Costs	Unit Price x Unit Sales	Total Selling Costs +(Unit costs x Unit Sales	Gross Revenue – Gross Costs
Book 2								
Book 3								
		Sum = Total Sales				Total Revenue = Sum of Revenue	Total Costs = Sum of Costs	Profits = Sum of Net Revenue

Figure 4.8. Layout for a budget planning model.

Taking Account

Another type of spreadsheet application is laying out simple accounts. The example given is a manufacturing account, which calculates the cost to a company of making things to sell. The cost of selling the finished items and the income received from their sale are not included; those appear on the trading account.

Although it looks complex, the manufacturing account shown in figure 4.9 is fairly simple. The hard work is identifying which costs should appear on the sheet, and that may be a job for an accountant. Once a list of costs has been agreed upon, they are totalled up.

At various stages, adjustments are made to take into account the difference in stocking levels between the start and end of the year. For example adding in the value of stock brought forward from last year and subtracting the value carried into next year. The cost is frequently sub-totalled, to give the total material cost for example. Overheads are listed separately in this model, only the total is carried into the main column: the formula held in cell C28 is B19 B28.

```
LA  SLOT=A1
CONTENTS=SMALL-TIME GADGETS LTD.
Wi TopL BotR Pos Cw  Bw Fmt  Opt
0  A1    C26       22  4  D0RB
0  .....................A.....................B.....................C
...1 >SMALL-TIME  GADGETS  LTD
...2  Manufacturing Account
...3  Year Ended 30th April
...4
...5                                                   £            £
...6 RAW MATERIALS
...7  Opening Stock                                            12510
...8  Purchases                                                71600
...9                                                        --------
..10                                                           84110
..11  Less Closing Stock                                       (9400)
..12                                                        --------
..13 TOTAL MATERIAL COSTS                                      74710
..14  Production Wages                                         37100
..15                                                        --------
..16 PRIME PRODUCTION COST                                    111810
..17
..18 OVERHEADS
..19  Packing Wages                           3900
..20  Packaging Materials                     4150
..21  Factory Rent                           16800
..22  Insurance                               1300
..23  Maintenance                             1100
..24  Services                                2300
..25  Ancilliary Wages                       22600
..26  Ancilliary Salaries                    13000
..27  Plant Depreciation                      6200
..28  Fixtures & Fittings                      350         71700
..29                                        --------      --------
..30                                                        183510
..31  Less Work in Progress                                  (1220)
..32                                                        --------
..33 GROSS PRODUCTION COSTS                                   182290
..34   Less Finished Stock                                   (11850)
..35                                                        --------
..36 COST OF GOODS TO
..37 TRADING ACCOUNT                                          170440
```

Figure 4.9. Manufacturing account.

Editing the Screen Window

In figure 4.9, the screen window has been redefined to give a narrower
left-hand border, by reducing the border width (Bw) to four. Each of the
three columns (Cw) has been widened to 22 characters, to leave enough
room for the text in column A. Finally, the number format for the whole
window has been set to D0RB, so any fractions are rounded to whole
numbers and minus numbers are shown in brackets, as they often are in
proper accounts. The new window definition is also shown in the
figure, in the status area. Once the display is correct, save the model
using a name like 'GADGET'.

The Printer Window

The PRINT command makes use of a window known as P0 in exactly the same way as the screen uses window zero. So the way a model prints can be changed by editing the printer window definition.

To print on paper exactly what is shown on the screen, this printer window definition P0 has to tally with the screen window zero. The easiest way to do this is to get a copy of the current screen window definition onto the Edit line by pressing f1 in the usual way:

```
f1
Window?
0
Wi TopL  BotR  Pos Cw  Bw Fmt  Opt
0  A1    C26       22  4  D0RB
```

Edit the window number Wi from zero to P0, and press RETURN. This method can be used any time when the printer window should exactly duplicate the screen window:

```
Wi TopL  BotR  Pos Cw  Bw Fmt  Opt
P0 A1    C26       22  4  D0RB
```

This technique is adequate for printing the contents of the 'HOUSE' and 'BEACH' models, but the 'GADGET' sheet is too large to be shown on-screen all at once. Like the screen, a printer is of fixed width. Only three columns of 'GADGET' can be shown on an 80–column printer; four columns each of 22 characters would be too wide. However, a printer can print an unlimited number of rows, so the printer window can be edited like this, to include extra rows:

```
f1
Window?
P0
Wi TopL  BotR  Pos Cw  Bw Fmt  Opt
P0 A1    C37       22  4  D0RB
```

Only the BotR co-ordinate has changed. This makes the printer window much larger than the screen window.

The whole model can be printed out by entering the PRINT command at the '=>' Command mode prompt. The final hard copy should look like figure 4.9, though of course the status area will not be printed at the head of the sheet.

5 : Using the Model
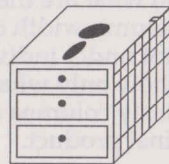
On-Screen Presentation

Imagine using a telescope to study a distant hillside. The single farmstead can easily be seen and each of its buildings. If a new town were built on the hill, the telescope would show only a few of the houses. Of course, moving the telescope, turning it left or right, up or down, would allow any group of buildings to be studied, but only a few at a time.

In ViewSheet, the screen is like a telescope offering a restricted view of part of the sheet. The telescope can be pointed at any part of the model, but if the model grows to be large and complicated, the overall plan of the sheet is hard to grasp. The important bits of the model can't all be on the screen at once.

Multiple Screen Windows

ViewSheet offers a sophisticated solution to this problem. It allows the model to be viewed through up to 10 separate telescopes or *windows*, each of them pointed at a different part of the sheet. This is the meaning of the window number Wi in the window definition: the windows are numbered zero to nine.

Window zero starts as the only window, and the view through it covers the whole screen. If window zero is shrunk to fill only a part of the screen, then there is room to use window one to look at another part of the model. This way, several of the most important bits of the model can be seen at once, even if they are at opposite ends of the spreadsheet.

It's important to realise that a complicated multi-window arrangement can sometimes be more of a hindrance than a help. The separate windows slow down movement across the sheet. It's often best to set up the sheet using the full-screen window first, and then arrange the windows you want afterwards.

So what are they for then? Using different windows is the only way the column width can be varied in ViewSheet; the columns themselves are not under individual control. Also, the windows can be arranged to show only what is necessary, for example, data and answers. All those nasty columns of manipulation in between need not be shown in the final product.

Working with windows is the most difficult bit of ViewSheet to grasp, so don't worry if it seems difficult; everyone is foxed for a while.

Planning Again

It's a good idea to roughly plan out the display on paper first; sketch in the windows, number them and name the cells at each corner of each window. Without doing this, it is very difficult to visualise the final display as it is being created.

There are a couple of points to note when planning a multi-window layout. The first is that the total width of the windows shown on the screen can't exceed 79 characters, including the widths of the dotted borders. Remember too that ViewSheet spaces all the columns within each window one character apart, and spaces the windows one apart. Therefore two windows each with three columns all of width seven, take up 48 characters, not 42, plus the width of the borders. Similarly, a single blank line is left between windows down the screen, and this has to be allowed for when counting the total number of lines in a display.

The sketch must take into account the screen mode; windowed displays set up for one screen mode can't be used in another. Modes 0 and 3 are generally the most useful, because they display the largest number of cells across the screen. OverView uses the *WIDE command to increase the width to up to 106 characters across the screen. This can be useful when setting up windowed displays, but that display layout can't then be used on machines without OverView.

Figure 5.1 shows a sketch that might be used with the 'GADGET' manufacturing account model from Chapter Four. The most important point to note is that only the first column needs to be wide as it contains text describing the account entries - the two remaining columns are only numbers, and so the can be much narrower.
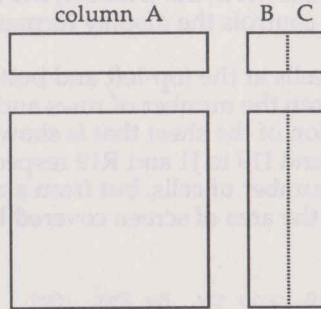
Figure 5.1. Sketched layout for the 'GADGET' model.

There are four windows shown on the sketch in figure 5.1. The intention is that any working on the sheet will be done in the lower two windows, while the top two always remain fixed on screen, at the top of the columns to show the headings. Of course, proper column headings could be used, but for long labels or descriptions they can be clumsy.

Editing Window 0

The first task in setting up any multi-window display is shrinking the main window to allow room for other windows. Load the 'GADGET' model and change to the chosen mode by using the MODE command:

```
MODE 3
```

Changing mode, or loading the model in the wrong screen mode always resets the windows to their normal, default state. Window zero fills the entire screen with columns seven characters wide, that is column A to column I in mode 3.

To change window zero to the size shown in the sketch, the window definition must be edited. This is achieved by pressing the EDIT WINDOW function (key f1), then enter '0' at the 'Window?' prompt. The normal, or default, window zero definition in mode 3 looks like this:

```
f1
Window?
0
Wi TopL  BotR  Pos Cw  Bw Fmt  Opt
0  A1    I19        7   7  FRM
```

Cw is the width of cells shown in the window, Bw is the width of the dotted left border. Fmt controls the display format of values in the cells.

TopL and BotR list the cells at the top-left and bottom-right of the window; this controls both the number of rows and columns shown on the screen, and the section of the sheet that is shown. If TopL and BotR were changed from A1 and I19 to J1 and R19 respectively, the screen would show the same number of cells, but from a different area of the spreadsheet. To shrink the area of screen covered by the window, BotR could be edited to A5:

```
Wi  TopL  BotR  Pos  Cw  Bw  Fmt   Opt
0   A1    A5         30  7   D0RB
```

The window would show only five cells, in the top-left corner of the screen. The window should show the title labels of the sheet, but just like the full screen window, the tiny window can be scrolled to show any area of the sheet. Try it, but to avoid confusion remember to go back to cell A1 before adding a second window.

Adding a Second Window

The reduced size of window zero leaves room for another window. To add the second, edit window one:

```
f1
Window?
1
Wi  TopL  BotR  Pos Cw  Bw  Fmt   Opt
1   A1    A1        0   0   FRM   0
```

The final 'O' under Opt indicates the window is off; deleting the O switches the window on. Alter the definition like this:

```
Wi  TopL  BotR  Pos Cw  Bw  Fmt   Opt
1   B1    C5        R0  10  4   D0RB
```

Note that the TopL and BotR cells have been changed, so that the window displays cells adjacent to those in window zero. The other new bit is the Pos 'R0'. This shows the window should be displayed to the *right of window zero*. The other possibility is 'B0'; then window one would be shown *below window zero*. The position must always refer to a lower-numbered window, so window one must always refer to

window zero, whereas window three could refer to any of windows zero, one or two.

When RETURN has been pressed, window one is displayed alongside the new, smaller window zero. The dotted borders make it clear where the edges of each window are, and the cell cursor is visible in window zero. Try moving the cell cursor around within the window; the window scrolls just like it did when it covered the whole screen, without affecting the other window. At any time, the cell cursor can be switched to the other window by pressing the function key f2, which is marked NEXT WINDOW on the keystrip.

Opt can be used to switch off the window (with O), and it can also switch off the dotted borders at the top and left-hand sides. T removes the dotted line across the top of the columns and S removes the left border. TS removes both, but this makes it impossible to tell which cell is which, because to add to the confusion the column and row names are removed too! However, cell names like C10 can still be used in formulae by pointing with the sheet cursor and pressing SHIFT–COPY. Turn off just the side border for window one:

```
Wi TopL  BotR  Pos Cw  Bw Fmt   Opt
1  B1    C5    R0  10  4  D0RB   S
```

The sheet should now look like figure 5.2. Note that with the increased width allowed for column A in window one, the whole of the text in cells A1 to A3 can now be seen. Previously, only the first 22 characters of the text could be shown. Now there is room for 30 characters, because less space is taken up by columns B and C.

```
A   SLOT=B1
CONTENTS=*Blank*


0   ...........................A..........B..........C
...1 SMALL-TIME GADGETS LTD.    ▷
...2 Manufacturing Account
...3 Year Ended 30th April 1987
...4
...5                              £       £
```

Figure 5.2. The 'GADGET' model through two windows.

A Four Window Display

The lower two windows in the layout sketch in figure 5.1 can be added in the same way as window one. The window definitions are as follows for window two:

Wi	TopL	BotR	Pos	Cw	Bw	Fmt	Opt
2	A6	A18	B0	30	4	D0RB	VT

and for window three:

Wi	TopL	BotR	Pos	Cw	Bw	Fmt	Opt
3	B6	C18	R2	10	4	D0RB	VTS

Note window two must be positioned below window zero. Window three could equally well be below window one, or to the right of window two. Once these have been set up, the sheet should look like figure 5.3.

There is one new feature in these last two window definitions, 'V' is set under Opt. This V stands for *vertical scrolling*, and means that when the window with the cell cursor scrolls vertically, all the others with V set also scroll to keep in step. This keeps the text in column A in line with the right figures in columns B and C.

```
A   SLOT=B6
CONTENTS=*Blank*

0  ..............................A .........B.........C
...1 SMALL-TIME GADGETS LTD.
...2 Manufacturing Account
...3 Year Ended 30th April 1987
...4
...5                                          £       £

...6 RAW MATERIALS                   >
...7    Opening Stock                              12510
...8    Purchases                                  71600
...9
..10                                               84110
..11    Less Closing Stock                         (9400)
..12                                             --------
..13 TOTAL MATERIAL COSTS                          74710
..14    Production Wages                           37100
..15                                             --------
..16 PRIME PRODUCTION COST                        111810
..17
..18 OVERHEADS
```

Figure 5.3. The completed four-window 'GADGET' display.

It is possible to set 'H' under Opt too, this links together windows that scroll horizontally. One common layout is shown in figure 5.4, with the H and V options set for various windows. While working in window three, the text headings in windows one and two keep in step whichever way window three is moved. Windows two and three are linked vertically, windows one and three horizontally. Figure 5.4 also shows the range of S and T options, and the way they affect the top and side borders.
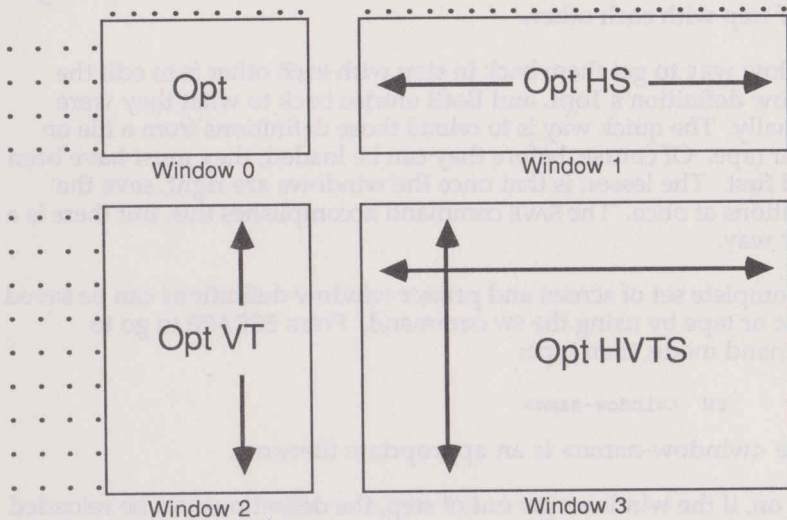
Figure 5.4. A common four window scrolling scheme.

When working on a sheet like this, it's usually most convenient to have the cell cursor in the largest window - in this case window three. Sometimes it is necessary to go to another window, and the NEXT WINDOW function (key f2) should be used for this. It cycles through the active windows; press this key once to go from window zero to one, then again to go from one to two, then two to three, then back to one again and so on.

Of course, any of the windows can be scrolled to bring any individual cell into view. However, it's seldom just one cell that needs to be looked

at, and the window may not be the right shape to show all of the area of interest. Using the NEXT WINDOW function key is usually better.

Saving and Loading Window Schemes

There is a problem with this linked scrolling. If one linked window reaches the edge of the sheet (ie, a cell in row 1 or column A is displayed) then it can't scroll any more. When this happens, the other windows it is linked to can continue to scroll. The result is that the windows can get out of step with each other.

The slow way to get them back in step with each other is to edit the window definition's TopL and BotR entries back to what they were originally. The quick way is to reload those definitions from a file on disc or tape. Of course, before they can be loaded, they must have been saved first. The lesson is that once the windows are right, save the definitions at once. The SAVE command accomplishes this, but there is a better way.

The complete set of screen and printer window definitions can be saved to disc or tape by using the SW command. Press ESCAPE to go to Command mode, then type:

        SW   <window-name>

where <window-name> is an appropriate filename.

Later on, if the windows get out of step, the definitions can be reloaded using a similar command LW:

        LW   <window-name>

Another result of LW is that it also resets the screen mode to whatever it was when the windows were saved. Loading the whole model with LOAD doesn't do this; it just forgets the windows if the screen is in the wrong mode.

Tape: The window filename can be up to 10 characters long. Try to give the file a name that shows it is a set of window definitions, 'GADGET-WI' for example.

Disc: The filename may have up to seven characters, plus drive and directory name if necessary. It is sensible to keep all

window files in a separate directory P, so they are distinct from the model files (kept in directory M). The window file could be called 'P.GADGET' perhaps. The reason for using P (for pane) is that ViewSpell uses W (for words, presumably) to store its dictionary files. W (for window) could be used, but would be very confusing.

ADFS, Net: Directory and filenames may be up to 10 characters long. Keeping the window files in a directory separate from the model files is good. If the models are in the current directory VSHEET, then a sub-directory called VSHEET.WINDOWS is probably best, so the window file could be called 'WINDOWS.GADGET'. Of course, the WINDOWS sub-directory has to be created with the command *CDIR before use.

It is important to realise that LOAD and SAVE deal with the whole model, including the window definitions, whereas LW and SW deal with just the window definitions. If only SW is used, the model itself isn't saved!

Chameleon Colours

A final aspect of on-screen presentation is the choice of colours for the display. In mode 7, the colours can't be changed, but in modes 0 to 6, any of the BBC micro's colours can be used. With a colour monitor or TV, the real colours are shown; on monochrome monitors and TVs, various shades of grey are displayed.

The stark white on black display can be improved by adding a coloured background to the sheet. The colours can be changed as follows; make sure ViewSheet is in Command mode, then hold down CTRL and press:

    S04000

This changes the background colour to a dark blue and is equivalent to the BASIC language statement, VDU 19,0,4,0,0,0. The foreground can be coloured in just the same way. Hold down CTRL and type:

    S73000

This makes the text yellow.

In these examples, the first CTRL–S is always the same. The second character can be CTRL–0 to change the background, or CTRL–7 for the text colour. The third character indicates the colour to change to as shown below:

CTRL–0	black
CTRL–1	red
CTRL–2	green
CTRL–3	yellow
CTRL–4	dark blue
CTRL–5	pink
CTRL–6	light blue
CTRL–7	white

The final three keystrokes are always CTRL–0. Generally the most legible and restful colour combinations are white, yellow or light blue on a dark blue background. Dark blue on a light blue background works well, but not in modes 3 or 6 because of the black bars between the lines of text.

These colours are not saved in the file with the model or window data, but have to be set every time the sheet is loaded, or whenever the screen mode is changed.

Looking Good on Paper

Chapter 4 introduced the idea of the printer window. In just the same way that multiple screen windows can be set up, several different printer windows can be set up too. This allows the hard copy to have many of the features of the screen layout, with varied column and border widths. Just as on the screen, only the most important, selected parts of the model need be shown.

The printer windows also allow some extra features. First, the maximum width of the windows may add up to as many as 255 characters, printer allowing. Second, the contents of individual windows may be printed out in bold face, or underlined.

Printing the Screen Windows

Copying the layout of the screen windows on paper is very
straightforward, just an extension of the technique used earlier to copy
the definition of screen window zero into printer window zero. Once
again using the 'GADGET' model - with four windows active on screen -
copy screen window zero into printer window P0. Window one also has
to be edited into the P1 definition, window two into P2 and window
three into P3.

Start by editing window zero:

```
f1
Window?
0
Wi  TopL  BotR  Pos Cw  Bw Fmt   Opt
0   A1    A5        30  4  D0RB
```

Change this to:

```
Wi  TopL  BotR  Pos Cw  Bw Fmt   Opt
P0  A1    A5        30  4  D0RB  TS
```

Similarly, window P1 reproduces window one:

```
Wi  TopL  BotR  Pos Cw  Bw Fmt   Opt
P1  B1    C5    R0  10  4  D0RB  TS
```

Note that for both these windows, both the top and side borders are
suppressed by setting options T and S. Obviously, there is no need to set
the scrolling options H and V for printer windows; if they are set, then
ViewSheet ignores them.

The two lower printer windows can be set in the same way. Remember
that they both need to extend further down the sheet to row 38, so that
the whole model is printed out:

```
Wi  TopL  BotR  Pos Cw  Bw Fmt   Opt
P2  A6    A38   B0  30  4  D0RB  TS
```

and window three:

```
Wi  TopL  BotR  Pos Cw  Bw Fmt   Opt
P3  B6    C38   R2  10  4  D0RB  TS
```

Again, all the borders are switched off to help tidy up the printout.

With this multi-window arrangement, it is worth checking the way that the model will be printed, before committing it to paper. Like VIEW, ViewSheet has a SCREEN command to do this. Press ESCAPE to return to Command mode and at the prompt, type:

SCREEN

The model will be displayed on screen, using the printer windows. A check can be made and corrections done. Finally, if the printer windows are right save all the window definitions with the SW command. Then the model can be printed:

PRINT

The final printout should look like figure 5.5.

```
SMALL-TIME GADGETS LTD.
Manufacturing Account
Year Ended 30th April 1987

                                        £              £

RAW MATERIALS
  Opening Stock                                      12510
  Purchases                                          71600
                                                    _____
                                                     84110
  Less Closing Stock                                 (9400)
                                                    _____
TOTAL MATERIAL COSTS                                 74710
  Production Wages                                   37100
                                                    _____
PRIME PRODUCTION COST                               111810
OVERHEADS
  Packing Wages                         3900
  Packaging Materials                   4150
  Factory Rent                         16800
  Insurance                             1300
  Maintenance                           1100
  Services                              2300
  Ancilliary Wages                     22600
  Ancilliary Salaries                  13000
  Plant Depreciation                    6200
  Fixtures & Fittings                    350         71700
                                       _____     _____
                                                    183510
  Less Work in Progress                              (1220)
                                                    _____
GROSS PRODUCTION COSTS                              182290
  Less Finished Stock                               (11850)
                                                    _____
COST OF GOODS TO
TRADING ACCOUNT                                     170440
                                                    ========
```

Figure 5.5. Printout of the complete GADGET model.

Of course, the printer windows need not mirror the screen windows at all, they can be completely different if necessary. Remember that when

the SW command is used the printer windows are saved too. And
although changing screen mode resets all the screen windows to the
default single full-screen window, the printer windows are left
unaffected by ViewSheet.

Wider Layouts

The printer windows can be up to 255 characters across overall,
although an individual window can't be more that 253 characters wide.
The screen is a maximum of 80 characters wide, or 106 characters with
OverView. If a wide-carriage printer is available, then it may be useful
to arrange the printer windows so that their overall width exceeds the
width of the screen.

But be wary of doing this without a wide printer. It is difficult to add up
the total width of the windows in a multi-window scheme. The print-
out will be spoilt if the printer windows' total width is greater than the
printer can deal with. In this case, the last characters at the end of every
line 'wrap around' and print on the next line of the paper. Thus each
row of the model is spread over two lines of paper, and the neat tabular
layout is ruined. The PAGE preview program described next checks
whether the printer windows are too wide for the printer's capabilities.
The listing is at the end of this chapter on page 67.

If an extra-wide layout is set up, the SCREEN command will no longer
show a sensible picture of the sheet before printing it. This can happen
even with an 80-column printer, if the size of the model and lack of
memory make it necessary to work in a 40-column screen mode. The
PAGE program may be used as a check in this case too.

Previewing the Page

The PAGE program at the end of the chaper first appeared in the July
1987 issue of Acorn User magazine.

In order to use PAGE, type in the BASIC program listing starting on page
67 and save it. Don't use the filename 'PAGE'; use 'PAGEsrc' or something
similar. Remember to go into BASIC first.

The constant 'gxr' should be set to TRUE in line 60 if the Acorn Graphics
Extension ROM (GXR) is fitted to the micro, or if it is a Master series

computer (where the GXR is part of the operating system). If the final page preview program is to be used on machines without the ROM, then leave 'gxr' as FALSE.

After saving the BASIC program, run it. If there are no errors, the program will assemble and automatically save the machine code file 'PAGE'. This file is a VIEW family printer driver. In ViewSheet Command mode, it can be loaded by typing:

```
PRINTER PAGE
```

Once loaded, PAGE will work in any of the graphics modes 0, 1, 4 or 5, on a BBC B, or their shadow equivalents on a BBC B+, Master or Compact (modes 128, 129, 132, 133). If the display mode is either 16 colour, text-only or teletext (modes 2, 3, 6 or 7, 130, 131, 134 or 135), then it must be changed to one of the graphics modes; it doesn't matter which one. Using mode 3 is usual when working on a model, so mode 4 is probably the most convenient to change to:

```
MODE 4
```

Now the model can be previewed, by using the PRINT command as normal. Type in:

```
PRINT
```

PAGE draws the outline of a piece of 'paper' on the screen, then adds the words and numbers in the form of small black bars rather than individual characters. Obviously the model can't be read, but the overall proportions of the paper page can be judged. It is possible to check the window layout without actually printing out a copy. If the windows are too wide to fit the paper, this is clearly shown on-screen.

With PAGE, when one piece of paper is filled up, the program pauses; move on to the next sheet by pressing the space bar when the 'Next page...' prompt is displayed. If this happens, it means that ViewSheet can't fit the model onto a single sheet. Because ViewSheet doesn't have any facility for pausing during printing (there is no equivalent of VIEW's SHEETS command), this shows that the model can only be printed on continuous stationery.

Pressing ESCAPE at any time returns the PAGE program safely to ViewSheet Command mode.

By using PAGE to preview the layout of the printer windows, these can be checked before they are printed out. If wrong, they can be edited, and PAGE used again. Once the windows are correct, PAGE can be removed, and ViewSheet can be prepared for normal printing, by entering:

PRINTER

Matching PAGE to Different Stationery

As it stands, PAGE will display text as it would appear in a pica (10 characters per inch) font on A4 paper. Check the program out by previewing a multi-window scheme less than 80 columns wide, for example the four window layout used for the 'GADGET' model.

When the routine is working properly, versions of PAGE may quite easily be customised for wider printers, different stationery or for another type pitch. To assemble a new version, changes can be made to the values given to the constants *lines* and *chars* in lines 70 and 80. These denote the number of lines per page and the number of characters across the whole width of the paper.

As an example, A4 paper used sideways (or *landscape* as it is called; upright is *portrait*) is about 11.7-inches across by 8.25-inches down. That leaves room on the sheet for 117 characters across the paper (11.7-inches at 10 characters per inch), and 49 lines down the paper (8.25-inches at six lines per inch). Thus, lines 70 and 80 could be changed to:

```
70 lines = 49
80 chars = 117
```

The dimensions of other common stationery are given in figure 5.6. The maximum size the screen can accommodate is 82 lines of 160 characters. If the paper is too big, the program still works but only the central portion of the paper is shown.

	Lines	Width in Chars	
		pica	elite
9.5 x 11 cont.	66	85	102
14.5 x 11 cont.	66	135	162
A5 (portrait)	50	59	70
A5 (landscape)	35	82	99
A4 (portrait)	70	82	99
A4 (landscape)	49.5	117	140
A3 (portrait)	99	117	140
A3 (landscape)	70	165	198

Table assumes $\frac{1}{6}$ inch line spacing and $\frac{1}{10}$ (pica) or $\frac{1}{12}$ inch (elite) character spacing.

Figure 5.6. Common paper sizes.

When possible, it is also a good idea to incorporate some reference to the stationery size and pitch in the filename, for example 'A4Lpica'. This file-name can be changed in line 90 of the program.

Only lines 60, 70, 80 and 90 should ever require alteration when making up a new version of PAGE.

PAGE can also be used to check whether a set of printer windows are too wide for the printer. If the printer is capable of printing only 80 characters across, then make up a version of PAGE with chars set to 80:

```
80 chars = 80
```

If a model is previewed with this driver, then any text that goes off the 'paper' on screen would also wrap around on the printer and spoil the neat columnar layout.

Technical Features of PAGE

Whichever screen mode PAGE is used in, a screen page appears to have about the same proportions as the final sheet of paper. In fact, the picture is drawn slightly too wide, as the ratio of height to width of a cell (one character by one line) is 1.5 to 1 on screen, but 1.67 to 1 for a standard printer in pica mode. The page width is further exaggerated in elite or condensed mode, but in practice the distortion is acceptable.

If the gxr constant mentioned earlier is TRUE, then the program draws
each page faster. This is because the rectangle drawing routines (PLOT
codes 96 to 103) can replace more complex groups of lines and triangles.
It is worth noting that where possible, PAGE draws by inverting the
present screen colour. So whatever combination of text and graphic,
foreground and background colours is used, both the paper and the
characters on it will be visible.

This inverting of screen colours is the reason PAGE cannot work with
mode 2. In other graphics modes, the inverse of the background colour
is the foreground, but in mode 2, inverting the background results in a
flashing colour. The resulting display is unusable, and so the PAGE
driver checks for mode 2 along with the non-graphics modes.

Program Listing 5.1

```
10 REM VIEW PAGE driver source
20 REM by Graham Bell
30 REM for B/B+/M/C/E + VIEW family
40 REM (C) 1987 Acorn User
50 :
60 gxr = FALSE
70 lines = 70
80 chars = 82
90 name$ = "PAGE"
100 REM alter for different stationery - default is
    A4 Pica
110 :
120 osbyte = &FFF4
130 oswrch = &FFEE
140 osrdch = &FFE0
150 :
160 DIM code &FF
170 FOR I% = 0 TO &FF
180 I%?code = 0
190 NEXT
200 :
210 off = code - &400
220 :
230 FOR pass = 0 TO 3 STEP 3
240 P% = code
250 [        OPT pass
260          JMP output - off
270          JMP newpg - off
280          RTS
290          BRK
300          BRK
310          RTS
```

```
320 .char    BRK
330 .line    BRK
340          RTS
350 :
360 .error   BRK
370          OPT FNequb (128)
380          OPT FNequs ("Bad mode")
390          BRK
400 :
410 .wait    LDX #4
420          JSR text - off
430          JSR osrdch
440 .newpg   LDA #135
450          JSR osbyte
460          CPY #2
470          BCC modeok
480          CPY #4
490          BCC error
500          CPY #6
510          BCS error
520 .modeok  LDX #0
530          STX line - off
540 .newln   JSR text - off
550          STX count - off
560          STX count + 1 - off
570          RTS
580 :
590 .text    LDY data - off,X
600          LDA data + 1 - off,X
610          SEC
620          SBC data - off,X
630          TAX
640 .loop    LDA data - off,Y
650          JSR oswrch
660          INY
670          DEX
680          BNE loop
690          RTS
700 :
710 .output  STA char - off
720          TXA
730          PHA
740          TYA
750          PHA
760          LDA line - off
770          CMP #lines
780          BCC notpg
790          JSR wait - off
800 .notpg   LDA char - off
810          BMI return
820          LDX #1
830          CMP #13
840          BNE notln
```

```
 850         JSR newln - off
 860         INC line - off
 870         JMP return - off
 880 :
 890 .notln INX
 900         CMP #32
 910         BNE print
 920         INX
 930 .print JSR text - off
 940         SEC
 950         LDA count - off
 960         SBC #8
 970         STA count - off
 980         BCS return
 990         DEC count + 1 - off
1000 .return PLA
1010         TAY
1020         PLA
1030         TAX
1040         LDA char - off
1050         RTS
1060 :
1070 .data   OPT FNequb (strin0 - data)
1080         OPT FNequb (strin1 - data)
1090         OPT FNequb (strin2 - data)
1100         OPT FNequb (strin3 - data)
1110         OPT FNequb (strin4 - data)
1120         OPT FNequb (strin5 - data)
1130 :
1140 .strin0 OPT FNvdu (26)
1150         OPT FNvdu (FNcls(gxr))
1160         OPT FNorig(640-chars*4,496-lines*6)
1170         OPT FNplot(4, 0, 0)
1180         OPT FNpage(gxr)
1190         OPT FNplot(4,0,lines*12-5)
1200 .strin1 OPT FNplot(0, 0, -12)
1210 .strin2 OPT FNdot(gxr)
1220 .strin3 OPT FNplot(0, 8, 0)
1230 .strin4 OPT FNequs("Next page..")
1240 .strin5 BRK
1250 ]
1260 count = strin1 + 2
1270 NEXT
1280 :
1290 sum% = 0
1300 FOR I% = 0 TO &FF
1310 sum% = sum% + I%?code
1320 NEXT
1330 IF (sum% <> FNtotal (gxr)) AND (lines = 70) AND
     (chars = 82) THEN PRINT "Assembler error -
     please check listing": END
1340 :
```

```
1350 PROCoscli ("SAVE " + name$ + " " + STR$~ code +
     " +100 40C 400")
1360 END
1370 :
1380 DEF FNdot (graphic%)
1390 IF graphic% THEN GOTO 1460
1400 [       OPT pass
1410         OPT FNplot (2, 7, 0)
1420         OPT FNplot (0, -7, -7)
1430         OPT FNplot (2, 7, 0)
1440         OPT FNplot (0, 1, 7)
1450 ] = pass
1460 [       OPT pass
1470         OPT FNplot (98, 7, -7)
1480         OPT FNplot (0, 1, 7)
1490 ] = pass
1500 :
1510 DEF FNpage (graphic%)
1520 IF graphic% THEN GOTO 1580
1530 [       OPT pass
1540         OPT FNplot (4, chars * 8 - 1, 0)
1550         OPT FNplot (85, 0, lines * 12 - 1)
1560         OPT FNplot (85,chars*8-1, lines*12-1)
1570 ] = pass
1580 [       OPT pass
1590         OPT FNplot (102,chars*8-1,lines*12-1)
1600 ] = pass
1610 :
1620 DEF FNcls (graphic%)
1630 IF graphic% THEN = 12 ELSE = 16
1640 :
1650 DEF FNtotal (graphic%)
1660 IF graphic% THEN = &4681 ELSE = &4A49
1670 :
1680 DEF FNvdu (code%)
1690 = FNequb (code%)
1700 :
1710 DEF FNorig (x%, y%)
1720 pass = FNequb (29)
1730 pass = FNequw (x% AND &FFFF)
1740 = FNequw (y% AND &FFFF)
1750 :
1760 DEF FNplot (code%, x%, y%)
1770 pass = FNequb (25)
1780 pass = FNequb (code%)
1790 pass = FNequw (x% AND &FFFF)
1800 = FNequw (y% AND &FFFF)
1810 :
1820 DEF FNequb (byte%)
1830 ?P% = byte%
1840 P% = P% + 1
1850 = pass
1860 :
```
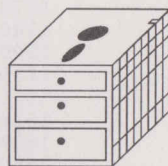
```
1870 DEF FNequw (word%)
1880 ?P% = word% MOD 256
1890 P%?1 = word% DIV 256
1900 P% = P% + 2
1910 = pass
1920 :
1930 DEF FNequs (string$)
1940 $P% = string$
1950 P% = P% + LEN string$
1960 = pass
1970 :
1980 DEF PROCoscli (string$)
1990 LOCAL X%, Y%
2000 DIM X% &FF
2010 Y% = X% DIV 256
2020 $X% = string$
2030 CALL &FFF7
2040 ENDPROC
```

Listing 5.1. PAGE preview driver.

6 : Printers and Drivers

Printer Drivers and ViewSheet

When the PRINT command is used in ViewSheet, the text and numbers on the sheet are usually sent directly to the printer, taking into account any printer windows that are active. This is straightforward, but it doesn't make good use of the capabilities of the printer. Most printers have a bold font, and can underline characters; many can print in italics and in many other different styles as well.

Unfortunately, problems arise because different types of printer need different *control codes* to use these special bold and underlined effects. Control codes are the printer's equivalent of VDU codes, and they can be used to change the style of the print. For example, the control code known as ESC 'E': in BASIC, this could be sent to the printer using:

```
VDU 2, 1, 27, 1, ASC "E", 3
```

On an Epson dot-matrix printer, these codes make the print style bold, but if sent to a Juki 6100 daisy-wheel printer, the text is underlined! In fact, by far the majority of printers used with BBC micros are of the dot-matrix type, and most of the modern ones are Epson-compatible (or nearly so). These are considered the 'standard' printer to use with the BBC micro.

A further difficulty is that most printers do not have the same character set as the BBC micro: for example, what appears as a hash character (#) on the keyboard and the screen, is often printed as a pound sign (£).

To overcome these problems, ViewSheet and other members of the VIEW family can be used with a *printer driver*. This is a program which manipulates the output from ViewSheet before sending it on to the printer. It also controls the use of bold and underline effects. *Highlights* are used to set the underline or bold type. When printing, the text, numbers and highlights are sent to the printer driver, and from there on

to the printer. It is the printer driver's job to translate each highlight into the correct control codes required by the printer.

Because different printers require different control codes, a unique printer driver is needed for each type of printer.

Using a Printer Driver

Any printer driver that is suitable for use with VIEW is also suitable for use with ViewSheet. If one is not available, the next section shows you how to create one. Alternatively, the program disc accompanying this book contains a simple driver for Epson-compatible printers. Once a suitable driver is available, then the driver must be loaded before starting to print anything out. In ViewSheet Command mode, enter:

        PRINTER FX4

to load a driver called 'FX4'. When it's been loaded, anything destined for the printer will be sent via the printer driver. Just use the PRINT command as normal.

ADFS, Net: Commonly, the driver is not in the current directory. It might be in a personal library directory for example, so the command could be:

        PRINTER &.MYLIBRARY.

Net only: On a network, the driver might be in a public library in the main root directory instead. This could be accessed by using:

        PRINTER $.LIBRARY.F

     & indicates the *user root directory*, the one selected with *DIR. $ is the main root directory, of which LIBRARY is usually a sub-directory. The & feature is only available from Acorn Level III or SJ Research Econet network fileservers

73

Assembling New Printer Drivers

There are five standard drivers supplied in the Acorn Printer Driver Generator package: 'JP101' for the Olivetti Spark Jet printer, 'FX4' for Epson FX80 and the many other Epson-compatible dot-matrix printers, 'JUKI4' for the Juki 1600 daisy-wheel, and 'FAC2' and 'FLO16' for the Facit 8105 and Ricoh Flowriter daisy-wheel printers. The Master series Welcome disc contains a similar selection of standard drivers. The 'FX4' driver is the equivalent of the 'EPSON' driver on the Master Compact Welcome disc.

If the printer you use isn't included as standard, then the Acorn Printer Driver Generator can be used to create a new driver. Usually, the most difficult job is understanding the printer manual!

The Printer Manual

Many printer manuals are examples of the worst type of technical literature. They are often dauntingly large and difficult to understand. Many are badly translated (frequently from Japanese), and the programming examples are usually in Microsoft BASIC, which is a very different language from BBC BASIC. Also, many printers have features so complicated that they are hardly ever used.

To create a new printer driver, only a few features of the printer need be looked up in the manual. The important ones to look out for are listed below:

> reset printer
> set/cancel underline
> set/cancel emphasised mode
> select character set

There is often a control code summary in the manual, and this usually contains enough information. The codes in the summary might be listed in a variety of ways. To understand this, it is important to realise that there are two ways of refering to a single character; as a character (letter, number or punctuation) and by its ASCII code (which is the number actually sent to the printer). For example, the number nine (character '9') can be referred to as ASCII code 57.

Often, the codes in the manual's summary are written out something like this:

ESC @	1B 40	27 64	reset printer	page 144
ESC 4	1B 34	27 52	set italics	page 68
ESC – n	1B 2D n	27 45 n	set/cancel underline	page 72

Each of the first three code columns gives the same information in a different form. The first might be termed *mnemonic* form, the second is in hexadecimal (numbers in base 16), the third in denary (ordinary base 10 numbers). When using the Printer Driver Generator, the mnemonic is the easiest form to work with, but if programming the printer in BASIC the decimal form is more convenient.

Most control code sequences begin with ESC, the code generated by the ESCAPE key, so they are often called *escape sequences*. The ESC mnemonic represents a special character (in fact ASCII code 27) that tells the printer to expect something special. The following code in the sequence nearly always represents an actual character. Even ESC 4 means the character '4' (ASCII character 52), not ASCII code 4.

The third bit of the underline sequence is listed as 'n'. This doesn't mean the character 'n' (ASCII code 110), but a number, or *parameter value*. What this number should be, can be found by looking up the details of that particular control sequence:

$$n = 0 \ldots\ldots\ldots \text{ ends underlining}$$
$$n = 1 \ldots\ldots\ldots \text{ begins underlining}$$

The parameter is sent to the printer as an ASCII code, so n = 0 means ASCII code 0, not the character '0' (which is ASCII code 48).

The Generator

When the Printer Driver Generator is run, it prompts you with a series of questions about what features you want in the new printer driver, and which control codes the printer requires to switch on and off each of these effects. These codes must be found in the relevant printer manual.

Printer initialisation is usually necessary with dot-matrix printers. The printer reset sequence (usually ESC @) is sometimes used to get rid of any odd printer settings left over from previous programs. More often

though, a special code is needed to allow pound signs to be printed. On an Epson FX80 or Canon printer, pounds are unavailable until the ESCAPE sequence ESC I 1 is used: after this, code 6 prints a pound sign.

To answer the printer initialisation question with ESC I 1, enter the following at the prompts:

```
Include printer initialisation?
Y
Give code sequence for initialising the printer :
ESC "I" 1
```

Notice how ESC is entered as a mnemonic, the I is enclosed in quotes because it represents a character, but the ASCII code 1 is entered as a number and without quotes.If several of these sequences are needed, they can simply be joined together, for example:

```
ESC "@" ESC "I" 1
```

Auto line-feed depends on the printer. The tiny DIP switches inside the printer are sometimes set so that printing RETURN (ASCII code 13) makes the paper feed up one line automatically. Other printers are set so RETURN and a line-feed (ASCII code 10) must be printed separately. One way of checking this out before running the Generator, is to enter:

```
*FX 6 10
```

Make sure the printer is connected, then press CTRL–B and type RETURN a few times. Finally press CTRL–C to switch the printer off again. Check whether the printer moved the paper up, and printed something like the following:

```
=>
=>
=>
```

If so, then the printer has automatic line-feed. If the printer has not moved the paper at all, but just printed one prompt perhaps a little darker than normal, then it has no automatic line-feed.

The Generator asks whether the printer does line-feeds automatically. If still in doubt, try Y as this is most common. If all the text is later printed on one line, then try again with the answer N.

Underlining is the effect given by highlight one. The bold effect is controlled by highlight two. The control codes for these effects should be entered, first to switch the effect on, then to switch it off. Bold is often called double-strike or emphasised printing, it means the same text is printed twice to make the type darker.

The dollar, hash and pound signs are often the source of difficulties. The Generator must be told the codes that print each one of these characters. One complication is that many dot-matrix printers now have more than one character set, and not all the characters are available in each set. There is no hash sign in the United Kingdom set on Epson printers. Which set the printer uses is again controlled by the DIP switches inside the printer.

Overall, if the DIP switch settings can be changed, it's best to have the printer use the American character set. With the US set, most characters are printable, but this may make it difficult to print the pound. If a single code to print a pound sign can't be found (like ASCII code 6 on an Epson FX80), then the following sequence might be useful:

```
ESC "R" 3 "#" ESC "R" 0
```

This allows an Epson RX80 printer to print a pound sign, by switching from the US character set to the UK set (ESC R 3), printing a hash (which comes out as a pound sign, because there is no hash in the UK character set), then switches back to the US set (ESC R 0). Hash is printed as normal, because it takes its usual place in the US character set.

Conversely, the printer might be set up to use the UK characters, in which case a hash sign is difficult to print. The following codes are the reverse of those above:

```
ESC "R" 0 "#" ESC "R" 3
```

They print a hash by switching temporarily to the US set. To get a pound sign, a normal hash is used.

None of the other features that the printer driver provides for VIEW can be used in ViewSheet. Since the output from a spreadsheet model is tabular, microspacing would ruin the neat layout. Italics, an alternative font, extra characters or exact spaces, subscripts and superscripts can't be used, because they all require the highlights to be redefined or the use of extended highlight sequences. VIEW uses the HT edit command for

this, but ViewSheet doesn't have any equivalent. A Printer Driver Generator is also included on the disc accompanying *VIEW: A Dabhand Guide*. This is also suitable for use with ViewSheet.

Line Spacing Problems

Two of the most common printing problems are that the text may be printed all on one line, ie, the printer never moves the paper up, or all the text is double line spaced.

If you are using a printer driver created with the Printer Driver Generator, then the driver is the source of the problems. The wrong answer was given to the auto line-feed question. To remedy this, rebuild the driver from scratch with the Printer Driver Generator, using the correct answer for the auto line-feed question.

If you're not using a printer driver, then the following makes sure that the printer moves the paper up:

> *FX 6

If the problem is double spacing, the following ensures that the paper moves up only one line at a time:

> *FX 6 10

Master series   The computer can be configured so that *FX 6 is
                unnecessary:

> *CONFIGURE NOIGNORE

has the same effect as a permanent *FX 6, and:

> *CONFIGURE IGNORE 10

is like *FX 6 10.

Whether or not the printer has auto line-feed is controlled by a tiny switch, often called a DIP switch, inside or on the back of the printer. Changing the setting should be covered in the printer manual. Most BBC micro software assumes that the printer has auto line-feed.

Highlights in Window Definitions

So how are underline and bold actually used? The highlight codes in
ViewSheet are not placed directly onto the sheet. Highlights one and
two are inserted into the printer window definitions, as options one and
two. Each highlight applies to all the cells in that window. In addition
to the TS options to prevent the top and side borders from printing, this
printer window definition has highlight one set:

```
LA  SLOT=B4
CONTENTS=Johnson
Wi  TopL  BotR  Pos  Cw  Bw  Fmt  Opt
P1  B4    B7    R0   12  7   FRM  TS1
```

Before use, the printer driver must be loaded with the PRINTER command.
Then the PRINT command may be used in the normal way, and the sheet
is printed out as follows:

```
Johnson
Jones
Lake
Maitland
```

In exactly the same way, highlight two can be used to print out the
window in bold face, or both highlights can be used together to print
bold and underlined text.

As can be seen, ViewSheet only underlines the actual contents of the
cells, and not the whole cell width. If:

```
Johnson
Jones
```

is needed, then the cells on the sheet must be padded out with spaces to
the correct length. ViewSheet will underline the spaces.

Sometimes when highlight one is set, short lines are printed where the
cells should be blank. This happens because the cells on the sheet are not
in fact blank; they contain a few spaces, and the printer tries to
underline the spaces. If the offending cell is checked by moving the cell
cursor onto it, the status area will often say:

```
LA  SLOT=B8
CONTENTS=
```

Cell B8 is not really empty. To clear this up, use the DELETE SLOT function
(key SHIFT–f9). The status area changes to:

```
A   SLOT=B8
CONTENTS=*BLANK*
```

and there are no spaces to underline.

Viewing Figures

Whatever method is used to print the model out, ViewSheet does not
offer the degree of page layout flexibility VIEW does. It can't set
margins at either side of the paper, can't leave space at the head and
foot of each page nor automatically number pages. In fact, if
continuous stationery is unavailable then any sheet spreading over
more that a single piece of paper can prove difficult to print out. There
is no easy way to make the printer pause between sheets.

Sometimes it would be useful to use a ViewSheet spreadsheet within a
larger document, for example an account in a financial report or
statistics in an experiment. But ViewSheet can't be used to create text!

The answer to both of these difficulties is to transfer the ViewSheet
model into VIEW. Then the wordprocessor can be used to incorporate
the sheet and extra text into a larger VIEW file. When it is printed out,
all the facilities of VIEW can be used to control the layout of each page.
However, the spreadsheet file created with the SAVE command is not a
text file, and it can't be read directly into VIEW.

To create a text file that can be read by VIEW, a technique using *SPOOL
may be used. After setting up any printer windows necessary, return to
Command mode. Then to create the text file, type the following:

```
*SPOOL  <text-filename>
SCREEN
*SPOOL
```

This technique has two disadvantages. First, the file contains a few
extra lines of text which must be deleted once it is read into VIEW.
Second, the highlights attached to printer windows can't be transferred,
and inserting them manually in VIEW can be tedious. To simplify the
transfer of ViewSheet models with or without highlights, into VIEW, the
VSXFER program can be used. This program is known as a *spooler*.

The VSXFER Transfer Spooler

To use the VSXFER program, type in the BASIC program in listing 1 at the end of the chapter. Save the program using the filename 'VSXFsrc'. Then run it.

The program incorporates a routine to check the assembly language, and this has to be correct before the assembled spooler is saved automatically under the name 'VSXFER'. This spooler pretends to be a ViewSheet printer driver, but instead of sending the sheet to the printer, it sends all the output to a file instead.

To use VSXFER, load it with the PRINTER command:

```
PRINTER VSXFER
```

then PRINT the sheet, exactly as if a printer were in use:

```
PRINT
```

The spooler program prompts for a name for the new text file:

```
Spool filename ? <text-filename> RETURN
```

Enter a name for the new text file, ViewSheet spools the model to that file, taking account of the printer windows.

If you need to interrupt the spooling process, then press ESCAPE, not BREAK. This ensures the spooler closes the spool file and hands back to ViewSheet gracefully. If BREAK is pressed, then the file may be left open, and this can cause difficulties later on.

The spooled version of a model contains a text 'translation' of the work that can be read or loaded directly into VIEW, with everything appearing in the file as it would on a printed copy, including any highlights.

If the spooled file is to be read into the middle of a larger VIEW document that uses extended highlights (sequences of several highlights used to control printer effects), then because ViewSheet can't be used with extended highlights, a little more work is needed. One option is to replace all instances of highlight two (bold) with the equivalent extended highlight.

This can only be done in VIEW version three or later, using the following VIEW command:

```
CHANGE ^* ^*^*^* 1 2 RETURN
```

Do this after placing markers one and two at the top and bottom of the spooled spreadsheet. In earlier versions of VIEW, the extended highlight facility has to be switched off temporarily. Place the stored edit commands:

```
HT 2 129
```

and also:

```
HT 2 130
```

respectively at the top and bottom of the spooled spreadsheet.

There is usually no need to worry about reformatting the text in VIEW. Providing that the top and side borders are removed (with T and S options in the printer window definition) it won't spoil the neat tabular layout of the spreadsheet. Each line of a borderless model spooled from ViewSheet starts with a space, so they aren't affected by formatting. The only real limitation is that VIEW can't use files with a line length greater than 132 characters, so make sure the overall printer window width is less that that.

A Modified ASCII Spooler

It is possible to modify the VSXFER spooler so that it does not include any highlights in the transferred file. This may be useful if a spreadsheet needs to be transferred into another wordprocessor, such as Wordwise or InterWord, which can't interpret the VIEW highlights.

Listing 2 at the end of the chapter shows the changes that must be made to Listing 1 to create the new version of the spooler. Save the modified BASIC program using a filename like 'ASCIsrc', then run it. If all is well, then it will assemble and automatically save a new spooler called 'ASCII'. In all respects, this resembles VSXFER, except that no highlights are put in the new text file.

To use this spooler, load it as if it were a printer driver:

```
PRINTER ASCII
```

then use the PRINT command to print the spreadsheet. The spooler prompts for a filename for the new file, then the sheet is spooled into that file. The resulting pure text file can be loaded into almost any wordprocessor, or even sent by electronic mail. It contains no control codes or highlights, just ordinary text and carriage returns.

Technical Details of VSXFER and ASCII

These two programs first appeared in Acorn User magazine, in July 1986. Both programs are compatible with cassette, disc or network filing systems, on any Acorn machine that can run ViewSheet.

It is worth noting that the highlights in any ViewSheet model, when printed, send the codes 128 and 129 to the printer driver just as they do by default in VIEW. Of course, these codes can be redefined in VIEW using the HT edit command, but this is impossible in ViewSheet. When stored in a VIEW file, these highlights are not stored as codes 128 and 129 or their redefined values, but always as codes 28 and 29. So to make the model readable by VIEW, the spooler translates the highlight code one from 128 to 28 and anything above 128 to 29.

The spooler programs both deal with any errors or ESCAPE by closing the text file and handing control back to ViewSheet in an orderly manner. If BREAK is pressed accidentally whilst either spooler programs is working, then with many filing systems, the text file may be left open. This doesn't become obvious until later, as an error message 'Open' may be displayed when an attempt is made to delete or overwrite the file. Alternatively, a 'Too many files open' message may appear.

An operating system command exists to close these incomplete files so they can be deleted. This is *CLOSE, it is available on BBC B+ micros and the Master series. It closes all the open files on the current filing system. However, BBC B micros with Disc Filing System 0.9 or 1.2 don't have this command. The *ViewSheet and ViewStore: A Dabhand Guide* program disc (see Appendix E) contains a *CLOSE utility for BBC B micro owners.

Program Listings

```
 10 REM VSXFER transfer driver source
 20 REM by Graham Bell
 30 REM for B/B+/M/C/E + VIEW family
 40 REM (C) Acorn User July 1986
 50 :
 60 brkv = &202
 70 osfind = &FFCE
 80 osbput = &FFD4
 90 osasci = &FFE3
100 osword = &FFF1
110 osbyte = &FFF4
120 :
130 DIM code &FF
140 FOR I% = 0 TO &FF
150 I%?code = 0
160 NEXT
170 :
180 off = code - &400
190 :
200 FOR pass = 0 TO 3 STEP 3
210 P% = code
220 [        OPT pass
230          JMP output - off
240          RTS
250 .hand    OPT FNequb (0)
260 .yreg    OPT FNequb (0)
270          JMP close - off
280          RTS
290 .nbrk    OPT FNequw (break - off)
300          RTS
310 :
320 .text    OPT FNequs (" ? emanelif loopS")
330          OPT FNequs (CHR$ 13 + CHR$15)
340 :
350 .block   OPT FNequw (flname - off)
360          OPT FNequb (&4FF - (flname- off))
370          OPT FNequb (33)
380          OPT FNequb (126)
390 :
400 .openfl TXA
410          PHA
420          TYA
430          PHA
440          LDY #18
450 .messag LDA text - off,Y
460          JSR osasci
470          DEY
480          BPL messag
490 :
```

```
 500              LDA #0
 510              LDX #(block - off) MOD 256
 520              LDY #(block - off) DIV 256
 530              JSR osword
 540              BCC noesc
 550 :
 560              LDA #124
 570              JSR osbyte
 580              BRK
 590              OPT FNequb (128)
 600              OPT FNequs ("Escape")
 610              BRK
 620 :
 630 .noesc   LDA #128
 640              LDX #(flname - off) MOD 256
 650              LDY #(flname - off) DIV 256
 660              JSR osfind
 670              AND #&FF
 680              BNE flopen
 690 :
 700              BRK
 710              OPT FNequb (129)
 720              OPT FNequs ("Can't open file")
 730              BRK
 740 :
 750 .flopen  STA hand - off
 760              JSR vector - off
 770              PLA
 780              TAY
 790              PLA
 800              TAX
 810              RTS
 820 :
 830 .close    LDY hand - off
 840 .clall    LDA #0
 850              JSR osfind
 860              STA hand - off
 870 :
 880 .vector  SEI
 890              LDA brkv
 900              LDX nbrk - off
 910              STA nbrk - off
 920              STX brkv
 930              LDA brkv + 1
 940              LDX nbrk + 1 - off
 950              STA nbrk + 1 - off
 960              STX brkv + 1
 970              CLI
 980              RTS
 990 :
1000 .break    PHA
1010              TXA
1020              PHA
```

VAV—G

85

```
1030          TYA
1040          PHA
1050          LDY #0
1060          JSR clall - off
1070          PLA
1080          TAY
1090          PLA
1100          TAX
1110          PLA
1120          JMP (brkv)
1130   :
1140   .output PHA
1150          LDA hand - off
1160          BNE isopen
1170          JSR openfl - off
1180   .isopen PLA
1190          PHP
1200          BPL print
1210          AND #127
1220          BEQ htone
1230          LDA #&01
1240   .htone  ORA #28
1250   .print  STY yreg - off
1260          LDY hand - off
1270          JSR osbput
1280          LDY yreg - off
1290          PLP
1300          BMI noprin
1310          JSR osasci
1320   .noprin RTS
1330   :
1340   .flname
1350   :
1360   ]
1370   NEXT
1380   :
1390   sum% = 0
1400   FOR I% = 0 TO &FF
1410   sum% = I%?code + sum%
1420   NEXT
1430   IF sum% <> &49A2 THEN PRINT "Assembler error -
       please check listing": END
1440   :
1450   PROCoscli ("SAVE VSXFER " + STR$~ code + " +100
       40C 400")
1460   END
1470   :
1480   :
1490   :
1500   DEF FNequb (byte%)
1510   ?P% = byte%
1520   P% = P% + 1
1530   = pass
```

```
1540 :
1550 DEF FNequw (word%)
1560 ?P% = word% MOD 256
1570 P%?1 = word% DIV 256
1580 P% = P% + 2
1590 = pass
1600 :
1610 DEF FNequs (string$)
1620 $P% = string$
1630 P% = P% + LEN string$
1640 = pass
1650 :
1660 DEF PROCoscli(string$)
1670 DIM X% &FF
1680 Y% = X% DIV 256
1690 $X% = string$
1700 CALL &FFF7
1710 ENDPROC
```
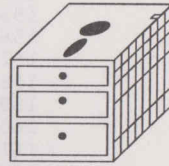
Listing 6.1. VSXFER transfer driver.

```
  10 REM ASCII spooler source
1190 BMI noprin
1200 :
1210 :
1220 :
1230 :
1240 :
1250          STY yreg - off
1290 :
1300 :
1430 IF sum% <> &46F1 THEN PRINT "Assembler error -
     please check listing": END
1450 PROCoscli ("SAVE ASCII " + STR$~ code + " +100
     40C 400")
```

Listing 6.2. Alterations to program 6.1 for ASCII spooler.

7 : Beginning with ViewStore

There are three fundamental types of database that can be used with a computer such as the BBC micro. The first is a store of information where the data can be divided only once, so that each section is considered as an indivisible packet. An example of this type of database is a Viewdata system such as Prestel. The data comes in pages, but you can't sub-divide those pages any further.

A more advanced type of database has a two-fold structure. The primary division of the database into records remains as above, but all the records can have sub-sections. Corresponding sections in each record contain the same type of information, so the structure within each record is the same. An address book has this type of structure; each entry (section) has several lines (sub-sections), one for the name, a couple for the address and one for the telephone number. This type of database is often described as a *flat file*, because it is two-dimensional. *ViewStore* manages this type of database.

The words you most often hear to describe the sections and sub-sections are *records* and *fields*. A flat file database is split into records, each of which has the same number of fields. In the address book, each name and address together form a single record. Each part of an entry forms a separate field, for example: name, address line one, address line two and telephone number fields.

The most complex type of databases on a micro are three-dimensional, and are described as *relational*. This type of database is characterised by having two or more flat file structures linked by sharing particular fields within each record. Relational databases lie at the heart of most business software, for example accounting and stock control systems. Here several sub-systems, such as purchasing and sales, each operate as separate flat files, recording the details of every transaction. But these flat files must all share the same 'number in stock' field.

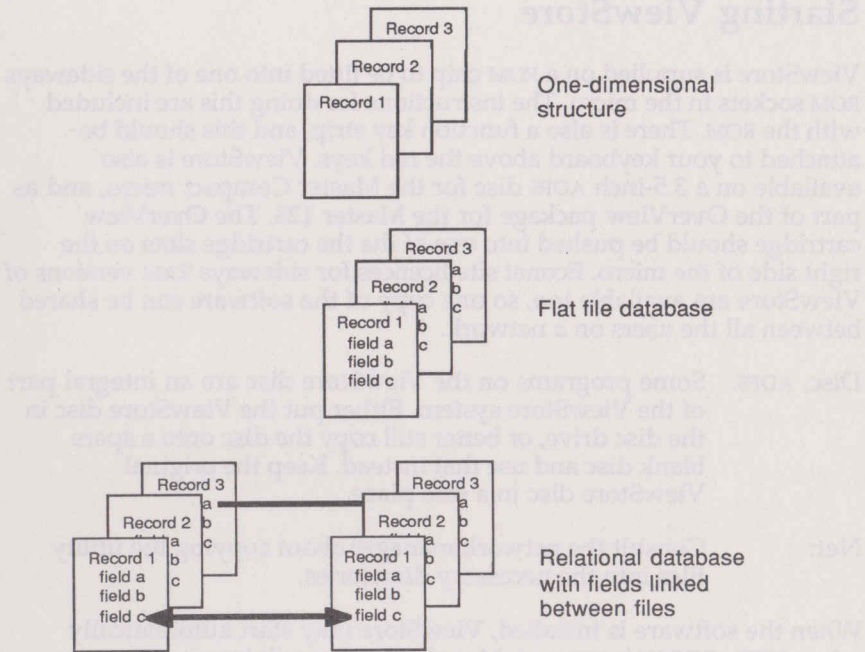Figure 7.1 summarises the three different database structures.

Figure 7.1 Three fundamental types of database.

The terms database and database manager can be confused, so let's clear that up. The database is just the store of information and the database manager is the program that helps to keep the whole thing organised. But a good manager can make all the difference between a valuable information resource and an attic full of junk.

The First Database

This section describes many of the basic features of the ViewStore database manager, and uses an example database on the *ViewSheet and ViewStore : A Dabhand Guide* programs disc.

Starting ViewStore

ViewStore is supplied on a ROM chip to be fitted into one of the sideways ROM sockets in the micro. The instructions for doing this are included with the ROM. There is also a function key strip, and this should be attached to your keyboard above the red keys. ViewStore is also available on a 3.5-inch ADFS disc for the Master Compact micro, and as part of the OverView package for the Master 128. The OverView cartridge should be pushed into one of the the cartridge slots on the right side of the micro. Econet site licences for sideways RAM versions of ViewStore are available too, so one copy of the software can be shared between all the users on a network.

Disc, ADFS: Some programs on the ViewStore disc are an integral part of the ViewStore system. Either put the ViewStore disc in the disc drive, or better still copy the disc onto a spare blank disc and use that instead. Keep the original ViewStore disc in a safe place.

Net: Consult the network manager about copying the utility files into the necessary directories.

When the software is installed, ViewStore may start automatically when CTRL–BREAK is pressed. More likely, you will have to type:

```
*STORE
```

at the usual '>' or '=>' prompt.

On entering ViewStore, the Command mode screen is displayed, showing the amount of free memory and the screen mode in use, for example:

```
ViewStore

Bytes free 29952
Editing No File
Screen mode 7

=>
```

The 'bytes free' figure shown is that for a 6502 second processor; with an ordinary BBC micro there will be fewer bytes free. On a Master series computer the figure will be different. The cursor flashes by the familiar '=>' prompt. When you see this prompt, ViewStore commands such as MODE 3 can be typed. So too can filing or operating system commands

like *CAT or *KEY. You can also choose another language, using a command such as *BASIC or *SHEET. Try typing in:

    *CAT

The disc catalogue should show at least some files in a directory 'U'. These files are the ViewStore utilities that can be used to manipulate the database.

Like VIEW and ViewSheet, ViewStore has two different modes, *Command mode* and *Data mode*. However, pressing ESCAPE won't immediately switch to the Data mode screen, because there isn't any data. An existing database has to be 'loaded' in before it can be examined. A database can't be created from scratch just by typing in data. To create even the tiniest database, a special SETUP utility program must be used. However, both the ViewStore utility disc and the *Viewsheet and ViewStore : A Dabhand Guide* programs disc contain a sample database, so a quick tour is in order!

Loading a Database

To load a database in ViewStore, the LOAD command is used. For example, to load the sample database on the ViewStore utility disc type:

    LOAD CARS

or to load the Dabhand Guide example, use:

    LOAD CREDIT

The CREDIT database will be used as an example, but exactly the same features can be seen with the CARS database.

The Command mode screen changes to show the name of the database in use, and consequently the amount of free memory is reduced. In fact, ViewStore doesn't actually read any of the data in at all, but it does load some *data about the data*. This 'database within a database' describes the format of the real database, so that ViewStore knows what to expect when it does read the data. At this stage, the screen displays something like:

    Bytes free 28956
    Editing D.CREDIT
    Format F.CREDIT
    Screen mode 7

91

Here, the file D.CREDIT is the database itself, and F.CREDIT' is the data about the data, telling ViewStore how D.CREDIT is arranged. D.CREDIT is called the *database file*, and F.CREDIT is its matching *format file*.

Data Mode Spreadsheet Display

Now the database is loaded in, we have to press ESCAPE to look at it and so switch to Data mode. The screen alters to show some of the data. Reading in the data may take a few moments, then the micro displays the message 'Reading' at the top left of the screen. Dots are printed to show progress as data is read in. When the data is read in, the screen looks something like figure 7.2.

```
L Space 9      Indexed by entry

Name of Creditor:

COMPANY.............ADDRESS 1..ADDRESS 2..ADDRESS 3..POSTCODE
Reliance Engineering Unit 4,    Wisbec.    Cambs.    CA2 1NN
Systematic Pipework  24 Spring  Ashbourne  Derbys.   DE17 1BF
Therm-Ace Heating    61 Railway Littleover Derbys.   DE5 2AB
Hi-Flo Pumps PLC.    Stator Hou Grove Road Derby     DE1 6FC
Therm-Ace Heating    61 Railway Littleover Derbys.   DE5 2AB
Systematic Pipework  24 Spring  Ashbourne  Derbys.   DE17 1BF
Bespoke Printers     1 Latimer  Derby                DE2 3FZ
Derby Journal Pubs.  Print Work 45-47 Nott Derby     DE4 5YU
Reliance Engineering Unit 4,    Wisbec.    Cambs.    CA2 1NN
Systematic Pipework  24 Spring  Ashbourne  Derbys.   DE17 1BF
Therm-Ace Heating    61 Railway Littleover Derbys.   DE5 2AB
```

Figure 7.2 Example CREDIT database, in Spreadsheet display.

At the top of the screen is a small status area showing the name of the database and other information. Below this is the data, arranged in columns and rows. Because this tabular arrangement is like a spreadsheet, it is called a *Spreadsheet display*.

The rows and columns of this display reflect the record and field structure of the database. Each record is shown on a single row across the screen; each field is a column. So a single *cell* such as the one picked out in white on the screen is a single field within one record. Below it is the matching field from the next record, and so on. The *field names* are shown at the top of each column; in figure 7.2 they are COMPANY, ADDRESS 1, ADDRESS 2 and so on.

As with a spreadsheet there may be some data in each record that can't be seen. The screen can only display a limited number of fields across

one row, and excess fields are 'lost' off the right-hand edge of the screen. You can scroll right to these fields.

To return to Command mode, simply press the ESCAPE key. This writes all the data back to the database file on disc before returning to the familiar Command screen.

Remember BREAK should not be pressed at any time in Data mode; always use ESCAPE to stop a process or return to the Command screen. Although BREAK usually just returns ViewStore to Command mode, sometimes it might corrupt the database file, and make it impossible to use in the future. Because this might happen by accident, a back-up copy of the database disc is vital. Ignoring the need for a back-up is courting disaster.

Net: Consult the network manager about how to make archive copies of the relevant database directories or files.

Card Display

An alternative to the Spreadsheet style display is the *Card display.* Just press CHANGE DISPLAY (function key f2) to swap to the Card display.

The screen changes to something resembling figure 7.3. On this display, each record looks like a card in a traditional card index box, or like a filled-in form. Two complete cards are shown in the figure. The field names are printed on each card, and the data for each field is printed beside the field name.

```
L Space 9        Indexed by entry
                             FICTIONAL CREDITORS

Name of Creditor:

   COMPANY Reliance Engineering

 ADDRESS 1 Unit 4, Da           INVOICE 100134
 ADDRESS 2 Wisbech                 DATE 16.4.86 <-- a record
 ADDRESS 3 Cambs.        STORES ITEM(S) Type 15 Jubilee Clip
  POSTCODE CA2 1NN              AMOUNT 209.1

   COMPANY Systematic Pipework

 ADDRESS 1 24 Spring            INVOICE DF309
 ADDRESS 2 Ashbourne               DATE 17.4.87 <-- a  eld
 ADDRESS 3 Derbys.       STORES ITEM(S) 1/2" Polypropylene P
  POSTCODE DE17 1BF             AMOUNT 46.0
```

Figure 7.3 Example CREDIT database, in Card display.

The differences between the two displays are not entirely cosmetic. The Spreadsheet display shows a large number of records, but only a few of the fields of each record are visible at any one time. In contrast, the Card display shows each record in full; all the fields are visible. But the price paid is that only a few records can be seen at one time. There are only two shown in figure 7.3, although more may be shown on the screen as ViewStore tries to fit on as many as possible.

To change back to the Spreadsheet display, just press CHANGE DISPLAY (function key f2) again. This key is always used to switch between the two types of display.

Just Browsing

Whichever display is used, one field of a single record is picked out in white on the screen. This white block is called the *field cursor*.

The field cursor marks the currently active field and record, and like all cursors it can be moved around using the cursor keys:

RIGHT	next field
LEFT	previous field
DOWN	next record
UP	previous record

Used with the SHIFT or CTRL keys, the cursor keys move the field cursor as follows:

SHIFT–RIGHT	last field of current record
SHIFT–LEFT	first field of current record
SHIFT–DOWN	next screenful of records
SHIFT–UP	previous screenful of records
CTRL–DOWN	end of database
CTRL–UP	first record of database

When in the Spreadsheet display, the fields 'hidden' beyond the right-hand edge of the screen can be inspected: just press the RIGHT cursor key until the field cursor is in the last complete field on the screen, then press RIGHT once more. The screen scrolls to the right to bring the next field into view. Further presses of RIGHT bring other fields onto the screen, until the field cursor reaches unused, blank fields. To return to the beginning of the current record, press SHIFT–LEFT.

Try applying the same principles to the Card display. Pressing RIGHT causes the field cursor to step through the fields of the current record, though the screen doesn't have to scroll at all because all the fields are already visible. Again, SHIFT–LEFT returns the white field cursor to the first field of the current record.

Don't worry if the disc drive occasionally whirrs into action as the field cursor is moved about. Databases can get very large, with the equivalent of several thousand cards worth of traditionally stored information. ViewStore obviously can't keep all the information in the BBC micro's small memory at once, so it doesn't try. It leaves most of the data on the disc, and just reads in the sections it needs at any one time, so occasionally you hear the whirring as the micro finds the data.

With the cursor keys, and SHIFT, browse up and down the database, using both the Card and Spreadsheet displays.

As you browse and get to know the structure, notice how some of the information displayed in the status area at the top of the screen changes. In the centre is the name of the database. This might say 'New cars: Specification and price' or 'Fictional Creditors', depending which example database is in use. Below the database name is a line that changes to give information about the field currently highlighted by the field cursor. This may be just the field name, which is also printed on each card on the Card display and at the head of each column of the Spreadsheet display. But it might be a prompt giving more explanation. For instance, the first field of each record of the CREDIT database is called 'Company', but the prompt which appears in the status area for this field is 'Name of creditor:'.

If the field cursor is moved beyond the last record of the database, or before the first record, the message 'End' appears in the status area.

Changing Data

So far, we have looked at our example database as an *archive*, - information that has been stored once–and–for–all, never to be changed. But real databases are not static; they are interactive and dynamic. So the data can be changed whenever necessary.

So how can a chunk of data be changed? Imagine that an accounting mistake is discovered, and that in reality the company owes Systematic

Pipework Ltd £57.59 for invoice number DF309, and not the £46.08 shown on the CREDIT database. Move the cursor to this field; it is the last field within the second record in the database. The screen should look like figure 7.4.

```
L Space 79      Indexed by entry

Outstanding Amount:

POSTCODE.STORES ITEM(S).......INVOICE....DATE.....AMOUNT.10
CA2 1NN   Type 15 Jubilee Clip 100134     16.4.86   209.15
DE17 1BF  1/2" Polypropylene P DF309      17.4.86
DE5 2AB   Moulded Elbows       00000001   17.4.86    11.97
DE5 2AB   Moulded T Joints     00000002   18.4.86     5.03
DE1 6FC   Spare Rotor          076105     20.4.86    21.52
DE5 2AB   Moulded Spigots      00000003   22.4.86    23.00
DE17 1BF  3/4" Nylon Hose      DG097      27.4.86    82.01
```

Figure 7.4 Altering an invoice total.

Under the first character of the field marked out in white is a flashing cursor; this is the *character cursor*, and it marks where alterations will occur if anything is typed at the keyboard.

To change £46.08 to £57.59, first ensure that the field cursor is on the correct field, and then simply type:

```
57.59
```

As this is typed, the new characters appear directly in the field. This new data replaces the original contents of the field as soon as it is typed, so all trace of the old and incorrect £46.08 is lost. So now the spreadsheet display shows this:

```
INVOICE....DATE.....AMOUNT
DF309      17.4.86   57.59
```

You don't have to press RETURN to 'enter' this new data. If you press RETURN, the field cursor just moves on to the beginning of the next field or next record.

Exactly the same process can be used on the ViewStore 'CARS' example database. Change the price of a Mini Mayfair from £3883 to £4259, by moving the cursor to the correct field (the PRICE field of the second record), and typing:

```
4259
```

When the change has been made, it is not necessarily recorded on disc immediately. But if the field cursor is moved to another part of the

database (perhaps by pressing CTRL–DOWN to go to the end of the data), then ViewStore writes the altered data back to the database file on disc, before reading in the new section of the database. In any case, the database file is always fully updated when ESCAPE is pressed to return to Command mode.

Changes can be made in exactly the same way on the Card display. Move the field cursor to the relevant field as before, and type in new information.

Editing Data

The changes made above completely replace the old information with the new, but changes are often merely modifications of the old data. In fact, the old data can be edited, rather than replaced.

The STORES ITEM(S) field of the second record contains '1/2" Polypropylene P'. This can be edited to contain 'Polythene' instead of 'Polypropylene', by typing over the letters which have to be changed. By pressing CTRL–RIGHT or CTRL-LEFT, move the flashing character cursor along the line until it is under the first 'p', then type:

```
thene
```

Notice how the new letters overwrite the old ones (similar to the way they do in VIEW). At this stage, the field should say:

```
1/2" Polythenelene P
```

with the flashing character cursor under the 'l'. The extra letters can be deleted using the DELETE CHARACTER function (press key f9).

As each extra letter is deleted, a new one appears at the end of the field. A ViewStore field can contain more data than is shown in the width allowed on the screen, and deleting a few letters at the beginning means more can be displayed at the end. A field like this can be *scrolled*, to bring the rest of the field contents into view by pressing CTRL–RIGHT until the end of the field is reached, or simply by one press of the END OF FIELD key (function key f5). There is a matching BEGINNING OF FIELD function too (key f4).

Extra spaces can be inserted into the field with INSERT CHARACTER (key f8), and the rest of the field right of the character cursor can be deleted if necessary with DELETE END OF FIELD (the f3 function key).

At the top of the screen in the status area:

         L Space 79

is shown. This means that there is room for 79 extra characters in that particular record. Editing the record will alter the space remaining, and adding lots of new data can result in no room being left in that record. If this does happen, then the computer beeps and the status area says:

         L Space 0

To add any more informaton, data will have to be deleted from other fields in the same record, freeing a bit of space. Perhaps something can be abbreviated?

Adding New Data

Putting a new record into the database is equivalent to writing out a new card in a traditional card index. But new electronic records can only be added at the back of ViewStore's pile of cards; they can't be slotted in at the appropriate place.

The following description uses the Spreadsheet display, so if the screen is in Card display mode, switch to the Spreadsheet display by pressing f2 (CHANGE DISPLAY). However, adding new data in Card mode is exactly the same as for spreadsheet mode. Similarly, if the CREDIT database from the *ViewSheet and ViewStore : A Dabhand Guide* programs disc is unavailable, adding a new record to the ViewStore CARS example database uses the same principles.

To add a new record, first move the field cursor to the last record at the bottom of the database by pressing CTRL–DOWN. The screen of the Dabhand Guide example database should look like figure 7.5; the ViewStore CARS example looks very similar, though the data is different. Now press the DOWN cursor key; the message 'End' appears in the status area at the top of the screen. The same message is displayed if the field cursor is moved above the first record too.

At this point the space remaining for the next record is very large.

```
L Space 29406 Indexed by entry

Name of Creditor
End
COMPANY..............ADDRESS 1..ADDRESS 2..ADDRESS 3..POSTCODE
Therm-Ace Heating     61 Railway Littleover Derbys.    DE5 2AB
```

Figure 7.5. Adding a new record to the database.

So how can new data be added. Imagine a new invoice must be added to the CREDIT database. On the blank line below the last record (or on the blank card after the last card in Card mode), type in the details of the new invoice. First, type in the name of the company. This goes in the first field. When the field cursor is on the COMPANY field, the prompt 'Name of Creditor' is shown in the status area, type in:

Huddersfield Brass Founders

There are three things to notice here. First, as the first letter is typed, there is a pause while the disc drive spins; ViewStore has to make sure there is room on the disc for the new record. Second, because the company name is longer than the width of the COMPANY field, the whole field scrolls as the name is typed in. Third, there is no need to press RETURN at the end of the name to enter the new data.

When the company name has been entered, move on to the next field using either the RIGHT cursor key or RETURN. This is the first of three address fields. Add the following address to the fields:

ADDRESS 1	Ideal Foundry
ADDRESS 2	Witherspoon Lane
ADDRESS 3	Huddersfield, W. Yorks
POSTCODE	BD21 7XT

Notice that some of the fields have extra prompts in the status area, which can explain what the contents of the field should be.

When the address fields are added, move on to the STORES ITEM(S) field, and use exactly the same procedure to add the following:

STORES ITEM(S)	Portsmouth Valves
INVOICE	I38715
DATE	5.6.87
AMOUNT	76.99

in the remaining fields. The screen scrolls to bring these fields into view. As the last piece of data is added, the screen should look like figure 7.6.

```
L Space 29244 Indexed by entry
                    FICTIONAL CREDITORS
Outstanding Amount:

ADDRESS 2..ADDRESS 3..POSTCODE.STORES ITEM(S).......INVOICE....DATE.....AMOUNT
Littleover Derbys.   DE5 2AB  Tap Washers     00000009  3.6.86     0.15
Witherspoo Huddersfie BD21 7XT Portsmouth Valves   I38715   5.6.86
```

Figure 7.6. Completing the new record.

Finally, when the new record is complete, press RETURN to go back to the beginning of the next record. As many new entries as necessary can be added.

At the end of the session, press ESCAPE to return to the Command screen. This makes sure that all the new data is saved onto the disc.

Why Use a Computer?

So far, we have used ViewStore just like a card index. The cards have been browsed through, and new cards added. But what advantages does the electronic database have over the paper-based methods of keeping such information?

Traditional cards are kept in a single order. A list of names and addresses are kept in name order, but not in order of the age of the person, nor grouped according to the addresses. A reference library often keeps more than one index system, so that books can be found by author, by title or by subject. Each book has three cards; one is kept with all the other cards sorted by author name, the second with another set of cards in order of title, and so on. And yet the information on each card is the same: author, title, subject, and the actual location in the library (perhaps a shelf number or Dewey classification).

But ViewStore does better, it can keep just one set of cards in several different orders! With the CREDIT database loaded, go to the Spreadsheet display as normal. The records are shown in the order in which they were originally typed in - that is what 'Indexed by entry' means on the top line of the screen.

To sort, or *index*, the cards into order, press f6 (the INDEX FIELD function). This produces a 'Fields:' prompt in the status area, showing the various *index fields* that the data can be sorted on. Picking the

COMPANY field would sort the records into alphabetical order of the company names, DATE would index the records in date order and so on. So to index the fields by company, press:

```
f6
Fields: COMPANY, INVOICE, DATE
Company
```

It doesn't matter whether the field name is in capital letters or lower case. The message 'Reading' is shown briefly while the records are read in from the disc in the right order, and after a few seconds, the screen should look like figure 7.7. The newly added record relating to the Huddersfield Brass Foundry is automatically put in its correct place.

The ViewStore CARS example database can be sorted into order by the same method; pressing f6 gives a list of the fields which can be used as index fields.

```
L Space 30     Indexed by COMPANY
                         FICTIONAL CREDITORS
Name of Creditor:

COMPANY..............ADDRESS 1..ADDRESS 2..ADDRESS 3..POSTCODE
Ball & Co            18 Leek Ro Ashbourne  Derbys.    DE17 3SD
Bespoke Printers     1 Latimer  Derby                 DE2 3FZ
Derby Journal Pubs.  Print Work 45-47 Nott Derby      DE4 5YU
Derby Journal Pubs.  Print Work 45-47 Nott Derby      DE4 5YU
Elite Stampings Ltd. 1 Longsdon Leek       Staffs.    ST14 1SR
Hi-Flo Pumps PLC.    Stator Hou Grove Road Derby      DE1 6FC
Hi-Flo Pumps PLC.    Stator Hou Grove Rd.  Derby      DE1 6FC
Huddersfield Brass F Ideal Foun Witherspoo Huddersfie BD21 7XT
Reliance Engineering Unit 4, Da Wisbech    Cambs.     CA2 1NN
Reliance Engineering Unit 4, Da Wisbech    Cambs.     CA2 1NN
S. & F. Motors Ltd.  Junction G Ten Mile W Derby      DE3 3SR
Systematic Pipework  24 Spring  Ashbourne  Derbys.    DE17 1BF
```

Figure 7.7 The records sorted by Company.

Of course, the order can be changed any number of times; press f6 again, and choose another field to base the index on. This is a fundamental difference between a traditional system and the computerised database; it means a single store of data can be used in many different ways.

When a database is sorted into an order this way, any record can be found very quickly, without searching through the entire database. For example while it is indexed by Company, to move directly to the new Huddersfield Brass Founders record, simply press the LOCATE function (key f7).

VAV—H

'Value?' is displayed in the status area, so type in at least part of the record to be located:

```
f7
Value?
Hudders
```

ViewStore redisplays the records, with the first one that matches the given value at the top of the screen.

One important point is that LOCATE only works on the current indexed field: if the records are in date order, then trying to locate 'Hudders' obviously won't work. Change the indexed field using the f6 key.

The example database can be arranged in order of company name, in date order, or by invoice number, but also in order of *entry*. To go back to the original order in which the records were typed in, press RETURN at the 'Fields:' prompt:

```
f6
Fields: COMPANY, INVOICE, DATE
```

It is best to re-order a database this way before adding any new data.

Getting Data Out

Another good reason for using a computer database is that the information in it can be used easily for other computer-based applications. For example, if it you needed to write letters to all the firms in the CREDIT database, then with traditional cards it would be necessary to read the cards and type each of the envelopes by hand—very time consuming!

But with ViewStore, there are a range of utilities to extract data from the database and use it in various ways. One of the easiest is one that prints labels for letters. Businesses often use sticky labels mounted on rolls of paper with sprocket holes so that they can be put through a printer. The labels can be printed automatically with the name and address of anyone on the database. This obviously saves a great deal of time.

To see how this is done, load the CREDIT database in the usual way. In Command mode, type the following:

```
MODE 6
UTILITY LABEL
```

This changes the screen mode to ensure there is enough memory free for the utility, and then loads the utility program. At this point, ViewStore may prompt:

```
Insert utility disc & hit a key
```

If so, put the ViewStore utility disc in the drive and press SPACE, then, when prompted, return the database disc to the drive and press SPACE.

The following prompts will appear:

```
Use select file (N,Y)? N
```

Answering 'Y' would print labels for only a selection of the companies in the database, but this selection hasn't been made yet, so the answer must be 'N'.

```
Screen or Printer (S,P)? S
```

Typing 'P' is possible, if there is a printer connected and ready to print. The answers below are for standard 1.5 by four-inch labels, but trial printing on ordinary paper would work too.

```
Label height (lines)? 8
Lines between each label? 1
```

The labels are 1.5inches, a total of nine lines of space on a normal printer.

```
Width of label? 35
```

There is room for up to 40 characters across a single four inch label. If the text on a label exceeds this maximum, then it will be shortened.

```
Characters between each label? 0
```

If there is more than one sticky label across the roll, then answer:

```
5
```

instead. The next question is:

```
Number of labels across the page? 1
```

If there is more than one label across the width of the roll, then type the number of labels across.

ViewStore now asks for the information to be put on the labels, one line at a time. Answer with the names of the fields from which to draw the data:

```
Line 1? COMPANY
Line 2? ADDRESS 1
Line 3? ADDRESS 2
Line 4? ADDRESS 3
Line 5? POSTCODE
Line 6? RETURN
```

After all the lines of the label have been specified, pressing RETURN on its own makes ViewStore move on. If the labels are being 'printed' to the screen, the labels are now displayed, but if the printer is in use, ViewStore now asks:

```
Alignment print (N,Y)? Y
```

ViewStore prints the first two labels, then pauses so they can be checked in case they have overrun the label, for example. If they are wrong, there is a chance to reprint the first two again. If they are right, then ViewStore continues to print the rest of the labels. The first couple of labels should look like figure 7.8. The use of this LABEL utility obviously lies in sending out batches of mail to people on a mailing list, the list being held in a ViewStore database.

```
Reliance Engineering Ltd.
Unit 4, Dampier Rd.
Wisbech
Cambs.
CA2 1NN
```

```
Systematic Pipework Ltd.
24 Spring Lane
Ashbourne
Derbys.
DE17 1BF
```

Figure 7.8 Two labels printed with the LABEL utility.

This LABEL procedure sums up many of the features of a computer database. For a very small group of simple records, then neither ViewStore nor any other database manager is worth the extra effort. But with a large or rapidly changing store of information which may be frequently accessed in a variety of ways, the database wins hands down. The ability of the database manager to sort the data using several different indexes, to locate any particular record quickly, and to extract data using utilities like the label printer far outweighs the cost of setting up and administering the electronic database.

8 : Creating a Database
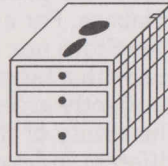
Building a new database is not a trivial task. Just collecting the information can be a time-consuming process, and then it has to be typed into the micro or written out in a manually-kept database. At least with a familiar card index, it is relatively clear what information should be collected, and it really doesn't matter how this data is arranged on each card. If new information comes along later, or if a vital facet of the data was left off, it can always be added to the cards at a later stage.

But with the ViewStore database the decisions made about the information to be stored, and how it is to be kept, are more important. Things can be modified in the light of experience, but major changes to the structure of the data are best avoided.

So time spent planning the structure of the database is time well spent. As an example, we'll consider the contruction of a database about articles in the microcomputer magazines, a sort of bibliography.

Working Backwards

Perversely, the most helpful way to think about the database is backwards! First ask yourself the question: 'What answers do I need from the database?' Think about how you want to use the information. This will lead to a list of types of information that have to be stored.

Taking the example article bibliography, think how a magazine index is used. The most common way is to look up an article whose title or author is known but the issue it appeared in has been forgotten. Rather than looking through a huge pile of magazines littering the floor, look the article up in the bibliography to find the issue date or volume number. So this means that the author's name, the title of the article and the magazine issue date form the core of each record in the database. Furthermore, it means that the Title and Author fields must be indexed fields, so that the LOCATE function (key f7) can be used on them. It might also be useful to have the page number in the database.

But what happens if a list of all the articles on a particular subject is needed. One option would be to find out if that subject appears in the titles of any articles. Or perhaps pick an author closely associated with that subject. But would either 'LOCATE Control' or 'LOCATE Telford' find all the articles about computer control applications? It is unlikely that they would all have the word ' Control' in their titles. Another way to do it is to use a series of keywords, perhaps with several words per article, describing the subject area of the article. So there is at least one other field to add to the proposed article index database.

We now have an author, the article title, the magazine name, a date and page reference, and a list of keywords for each article. Obviously, a single article relates to a single record in the database, but how should the information about each article relate to fields within each record?

It is usually clear how the data splits up, but some thought is needed. Should names be split into two fields, one for first name, one for second name? Alternatively, the name could be stored in a single field. But finding a name using LOCATE is usually done by surname, as it is more useful to use 'LOCATE Telford' than 'LOCATE Joe'. Is a list of people usually kept in alphabetical order of first name? So the choice is to have a single field (for example, 'Telford, Joe') or two separate fields. The principle to use in making this decision is to ask whether the two items of information would ever be wanted separately?

It is unlikely that in a magazine bibliography the first name of an author would be required, but if a name and address were used to write standard letters, then both surname and first name might be required. Consider the difference between the 'Joe Telford' on the envelope address, and the 'Dear Joe' at the head of the letter. A separate field to contain 'Mr', 'Mrs', 'Ms' or 'Dr' might also be helpful for mailing lists.

It is a good idea to jot down everything possible about each of the types of data to store in the database. Think about how long the data will be in each field; clearly the article title is going to be longer than the name of the author, on average, perhaps 20 or 30 characters. Figure 8.1 shows notes for the index of articles in the microcomputer press.

AUTHOR	In 'surname, forename' form; up to 20 characters long; must be indexed so articles can be listed in alphabetical order of author
TITLE	Article title; maybe 30 characters; must be indexed so articles can be listed in alphabetical order of title
MAGAZINE	Only needs one letter or a simple code (A=Acorn User, B=Byte etc)
DATE	Publication date; might range between 1/1/1970 and 31/12/1999 (day of month is not used for monthly magazines)
PAGE	A simple number between one and, say, 300 should cover most possibilities
KEYWORDS	Most important word first, so alphabetical order of keywords groups articles properly; must be indexed so articles can be found by keyword

Figure 8.1. Potential fields in article database.

Files, Files, Everywhere...

ViewStore uses a number of different types of file to manage its databases. We have already come across database files, format files and utility programs. It separates all these types of file by putting each in a different directory on the disc, so the format files are put in directory 'F' and utility program files in directory 'U'.

ViewStore keeps all the data for a database in a single file, in a directory called 'D'. For example, the CREDIT database on the programs disc accompanying this book is stored in a file called 'D.CREDIT'. There are also other types of file, index files for example, stored in directory 'I', show ViewStore which order the records should be in when indexed. There is one index file for each indexed field, so the 'CREDIT' database has three files in directory 'I'.

In addition to these data files, ViewStore uses two other directories ('S' and 'R') to save other types of file.

It is sensible to spread these files over whatever disc drives are available. The reason for this is that as new data is added to the database file, it grows. But if it grows too far, it may come up against either the end of the disc or the start of another file on the disc. With a DFS, this results in the 'Can't extend' error message being shown on screen. The aim of spreading the files around is to give each one the maximum amount of space to grow.

Files With DFS

The recommended arrangements for files, different types of disc drive and the DFS filing system are as follows:

Drive 0 only   Clearly, all the files have to be on one side of the floppy disc, although the ViewStore utilities in directory 'U' need not be copied onto the database disc if the database is expected to be large. When ViewStore can't find the utility program it needs, it prompts 'Insert Utility disc & hit a key', so the discs can be swapped. Remember you should use a back-up of the ViewStore utility disc, not the original.

Drives 0 and 1 With two single-sided drives, it's best to save drive 1 to hold a copy of the utility disc, and keep all the database files on a single disc in the other drive (drive 0).

Drives 0 and 2 A double-sided drive is probably best used by arranging the database file in directory 'D' on one side of the disc (drive 0) and keeping all the other files on the other side (which is drive 2). If the database is expected to grow very large, then it's better to keep the ViewStore utilities on a separate disc.

Drives 0 to 3   Two double-sided drives are the most convenient arrangement for ViewStore. The database file in directory 'D' can be kept on drive 0, while the files in directories 'F' and 'I' can be kept on the other side of the same disc, on drive 2. The other drive (drive 1) can be used to hold the utilities disc, the opposite side (drive 3) can be used to store temporary files that ViewStore creates in directory 'S'.

Files with ADFS Or Net

If you have an ADFS or network the best way to arrange the files are listed next:

One drive only   All the files have to be on a single disc, and with 640 kilobytes of space on a disc (assuming a large format), there is little point in not having the ViewStore utilities on there too.

Two drives   One floppy disc drive (called :0 or :4) can be used to store the data files in directories 'D', 'F' and 'I' and 'R', the other (:1 or :5) can hold the 'U' utilities and the 'S' temporary files.

Hard disc   Because a hard disc works faster than floppies, it's best to store all the files on the hard disc, in the same way you would with a floppy disc.

Net   Because the information in a database could be sensitive, it can be useful to have a separate user name on the network specifically for the database, so starting up the database may mean entering:

```
*I AM ACCOUNTSDB
```

for example. Consult the network manager about setting up a new user name and possibly protecting with a password.

Wherever they are stored, each of the directories 'D', 'F', 'I', 'S', 'R' and 'U' will have to be created with *CDIR before setting up the database, and the ViewStore utilities copied into directory 'U'. If necessary, these directories can be sub-directories, so the name of the 'CREDIT' database file on a hard disc could be '$.ACCOUNTS.VSTORE.D.CREDIT' while the matching format file could be '$.ACCOUNTS.VSTORE.F.CREDIT'. Figure 8.2 shows one possible directory tree for a single floppy disc under ADFS.
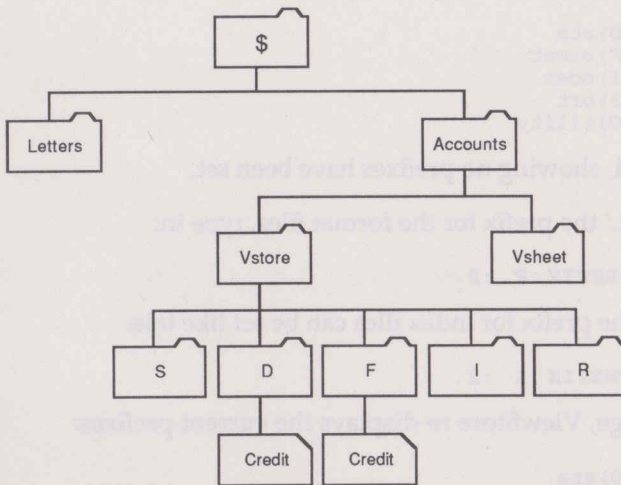
Figure 8.2. Possible ADFS directory tree.

Setting Up the Prefixes

When you have finished the basic planning, the database can be set up. Start up ViewStore in the usual way, or type:

    NEW

to tell ViewStore that no database need be loaded.

The first job is to specify where all the files are to be kept. This is done via a series of prefixes, which are added to the file-names to be used later. For format files, ViewStore assumes they are in directory 'F', but if the file is on drive 2, then the prefix ':2.' must be added. It isn't necessary to add any prefixes if all the files are being kept on a single drive, but they must be set if the files are spread across various drives.

Typing in the following:

    PREFIX

111

gives you a list of all the prefixes being used by ViewStore. Initially:

```
D)ata
F)ormat
I)ndex
S)ort
U)tility
```

is displayed, showing no prefixes have been set.

To make ':2.' the prefix for the format files, type in:

PREFIX F :2.

Similarly, the prefix for index files can be set like this:

PREFIX I :2.

At each stage, ViewStore re-displays the current prefixes:

```
D)ata
F)ormat  :2.
I)ndex   :2.
S)ort
U)tility
```

This shows that if ViewStore tries to load a database called 'CREDIT', it first tries to find the format file called ':2.F.CREDIT'.

The prefixes for the DFS file arrangement with two double-sided drives suggested earlier should look like this:

```
D)ata    :0.
F)ormat  :2.
I)ndex   :2.
S)ort    :3.
U)tility :1.
```

ADFS, Net: If the database directories are further down the directory tree, then the prefix can include longer directory names like 'VSTORE.'. In this case, ViewStore will search for a file called 'VSTORE.F.CREDIT'. However, it is easier to move through the directory tree (with the '*DIR VSTORE' command) so that the F directory is a direct sub-directory shown in the current catalogue.

Any files contained in directory 'R' automatically use the same prefix as the 'D' files.

Setting up the prefixes has to be done every time ViewStore is used, but it can be done automatically by setting up a !BOOT file as described in Chapter 16.

Setting Up the Bibliography Database

If there are already some files on the disc that will contain the database, first use *COMPACT. This makes sure the new database files are the last files on the disc, giving them as much space as possible to grow.

When the disc or discs have been prepared, the database files can be created with the ViewStore utility SETUP. From Command mode, type in the following:

```
UTILITY SETUP
```

ViewStore finds the 'SETUP' progam using the prefixes already set.

The initial prompt presented by the SETUP utility is:

```
Set up database or report (D,R)? D
```

Here, answer 'D'. In fact, pressing RETURN has the same effect; when ViewStore presents two alternative answers in brackets like D and R, then the first choice will be assumed if no answer is given. The construction of reports is covered in a later chapter.

The next prompt to appear is:

```
Filename of database BIBLIOG
```

The database file is to be called 'D.BIBLIOG'. Notice that ViewStore automatically assumes the 'D.', and that the prefix will also be added, so the file's complete could be ':0.D.BIBLIOG'. Obviously, the filename given must obey the rules for filenames on the current filing system. Next the filename for the format file has to be given:

```
Format filename (database filename)?
```

Here, just press RETURN if the format file is to be called the same as the database file itself (remember it is in a different directory). Otherwise, enter an acceptable name for the format file.

Reserving Database Disc Space

The next prompt is:

```
How many bytes to reserve?
```

Usually, no bytes need be reserved for the database file. It starts out with zero size, but grows as data is added to it. So the answer should be zero:

```
0
```

DFS Users Note

An initial size for the database file can be specified as additional insurance against the 'Can't extend' error message. If you can estimate the average number of characters per record and the approximate final number of records in the database guessed, then you can roughly work out the length of the database file. The size is:

1.1 x chars per record x total records

The bibliography might have perhaps 100 characters and 200 records, so the size is 100 x 200 x 1.1, which is 22,000. So enter:

```
22000
```

at the prompt given on screen.

Because of the way it works, the larger the initial size, the slower ViewStore loads the database. But as the database fills up with information, it will be loaded quicker! This procedure is only worthwhile if using DFS with drive 0 or drives 0 and 1 only.

Reserving Index Disc Space

While space doesn't need to be reserved for the database itself, to ensure that ViewStore operates satisfactorily space must be reserved for the index files.

Index files show which order the data should be displayed in when indexed on particular fields. For example, if the computer magazine articles are to be in order of author's name, then the data must be indexed on the AUTHOR field (with key f6). That causes ViewStore to check the order of the records in the relevant AUTHOR field index file.

In fact, ViewStore has two different types of index files: *updateable* ones which it builds as data is typed in, and *read-only* ones which are created after the whole database is complete. There is a limit to the number of updateable index files a database can have.

DFS, Net: There can only be four updateable index files, or three if you are going to use read-only indexes as well.

ADFS: There can be nine updateable indexes, or eight if read-only indexes are also to be used too.

In the bibliography example, there are three fields that are to be indexed fields, so three index files. But updating the index files as data is typed in slows ViewStore down, so only one will be used during data entry. The other two can be created later. The SETUP utility prompts for the number of indexes, and then for the expected number of records in the completed database:

```
Number of indexes? 1
Number of records? 200
```

The number of records is used to work out how much disc space to reserve for the index file. Always be generous and over-estimate the number of records to avoid problems as the database grows.

Finally, SETUP asks for the filename for each index file, and for the *key-size*. This is the part of the database field that is to be used for sorting the records into order; the number of letters taken into account when arranging the records alphabetically. Too short a key length and 'SMITH, Bruce' will not be differentiated from 'SMYTHE, Robin'. At least three characters are required in the key to split Smith and Smythe, and

eight to separate 'SMITH, Bruce' from 'SMITH, Sarah'. Remember the space counts as a character too. So :

```
Index 1:
Filename? AUTHOR
Keysize? 8
```

The key-size should always be the smallest that will differentiate all the records reliably. Too long a key just slows ViewStore down. If it is too short some of the records may be displayed out of order, though it does no other harm. Jot down the key-size chosen and the name of each index file; this information will be needed later.

If the indexed field were a number, then the key-size should be the number of characters in the largest values expected. For example, if the largest money value in a field were to be in the tens of thousands of pounds, then the key-size should be nine (23456.89, remembering to allow for the decimal point and places, and a possible minus sign too).

If the indexed field were a date, then the key-size should always be three, providing that dates are to be entered in the form 'dd.mm.yy' or 'mm.dd.yy'.

When the key-size is specified for all the index files, ViewStore creates the files necessary for the database, using the appropriate prefixes set up beforehand. Figure 8.3 shows all the answers given to set up the BIBLIOG database using DFS and a double-sided disc drive.

```
=>UTILITY SETUP
SETUP
Set up database or report (D,R)? D
Filename of database? BIBLIOG
Format filename (database name)?
How many bytes to reserve? 0
Number of indexes? 1
Number of records? 200
Index 1:
Filename? AUTHOR
Keysize? 8
Creating Format file
Creating index 1
Creating Database file
Database set up
=>
```

Figure 8.3 Creating the 'BIBLIOG' database.

116

9 : Customising the Database
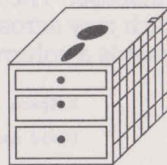
Loading a New Database

The new, empty database created as described in Chapter Eight, can be
loaded in just like any other by typing:

```
LOAD BIBLIOG
```

Notice that although the database file is in directory 'D' and the format
file is in directory 'F', there is no directory named in the load command.
This command also assumes that the prefixes are set up in the same way
as when the database was set up. If the database has only just been
created, then this will be true: if ViewStore has been switched off in the
meantime, then the prefixes may need resetting.

Press ESCAPE to switch to Data mode. The screen goes initially into the
Spreadsheet display, as shown in figure 9.1. The various fields are
merely numbered.

```
    L Space 29949 Indexed by entry


1..............2..............3..............4..............5.
```

Figure 9.1. Initial Spreadsheet display of blank 'Bibliog' database.

The display shown in figure 9.1 isn't encouraging. It *can* be used as a
database, but it's much better to customise the database to the specific
needs of the bibliography. To do this, the database has to be filled with a
bit of *data about the data*. This has to be put in two places: one called
the *database header* and the other is the *record format*.

The Record Format

To take a look at the record format, press f1. The display changes to
that in figure 9.2: it shows a blank record format. This is a mini-
database, containing information relating to fields in the main

VAV—I

database. The record format holds each field name and other details. Each row across the record format defines a single field of the database (that is a column of the database in Spreadsheet mode).

```
L Space 29949
                        Record format
Field name

Field name......Wid.T.S.D.Low limit..High limit.I.Key.Index name
```

Figure 9.2. Part of the record format.

Field Name

As discussed in Chapter Eight, the first field in the bibliography database should be the Author field. So under 'Field name', type in 'AUTHOR'. Entering this is just like typing information into the actual database. The second field should be 'TITLE', so enter this as the field name of the second field, in the second row. Names can be up to 15 characters long and you can insert spaces or numbers if you need to.

In the final database, the fields in Spreadsheet mode are always displayed in the order used in the record format. So it's a good idea to keep the most important fields, or those which will change the most, first on the list where they can be seen easily.

The term 'WID' is the *width* of space allowed on screen for the field. In Spreadsheet mode, a width of five would be a very narrow column of information, whereas a width of more than, say, 20 characters would be unnecessarily wide for the author's name. So enter '20' under width. Similarly, the article title field should be about 30 characters wide.

If the width of a field is narrower than the field name, only part of the name will be shown on the Spreadsheet mode display. The width can be zero. This means all the data is hidden; it can be neither seen nor changed. If the width is left blank, then ViewStore assumes a width of 18 characters.

Field Type

The following column of the record format, 'T', requires the *type* of the field. As the T field is filled in, 'Type: A/T,N,D/M' is displayed in the status area to suggest the available types. These are now described.

A   *Alphanumeric*. This allows both letters and numbers to be typed into the field, and is the most usual format. If you don't fill anything in here, an alphanumeric field is assumed. The Author field of the Bibliography database should be of this type.

T   *Text*. This also allows the field to contain any characters, but there is a subtle difference between alphanumeric and text fields. In alphanumeric fields, the information is considered as a whole; in text fields, each word or phrase can be considered separately. The Keyword and Title fields should be of this type so that articles can be found if you can remember only one or a few words of the title, or if you want to search through the list for all the articles relating to a certain keyword.

N   *Numeric*. Only numbers are allowed in N fields. They can be positive, negative, decimals, or a combination, for example, '-466.129'. Exponents such as 1.3E6 meaning 1,300,000 can't be used. The field width for a numeric field should always be at least large enough to show the longest expected number in full. Remember to count minus signs, decimal points and places. So -466.129 would require a width of eight. ViewStore won't accept letters in numeric fields and an error message of 'Not numeric' is shown in the status area at the top left of the screen.

D   *Date*. These must always be entered in the form 'dd.mm.yy', for example '12 07 87' meaning the 12th of July, 1987. The full stops don't have to be used; both '12   7   87' and '12/7/87' are acceptable. ViewStore prevents entry of silly dates like 31st of April, 1987 or 29th of February, 1987, while accepting 29th of February, 1988. One limitation on dates is that because only two characters may be used for the year, all dates must be from this century.

M   Dates can also be specified in American form: mm.dd.yy, so '12.7.87' means 7th December 1987. The British method of putting day first, then the month, are reversed in the American notation.

The next column in the record format is 'S'. This indicates whether the field contents should be allowed to *scroll* if there is more information than can be shown in the available width, say a 15-letter name in a field with a width of 10. Generally, text or alphanumeric fields should be allowed to scroll, numbers shouldn't. If S is left blank this is exactly what happens. A number longer than the field width can't be typed in, the computer just beeps. But there are times when an alphabetic field shouldn't scroll, perhaps to limit the text to a maximum number of characters. This might be the case with the Magazine field of the bibliography database. This could have a single character code for each magazine, and no scrolling and a width of one would limit entries to a single character.

The 'D' column is only relevant to numeric types: it should be left blank for other field types. D controls the number of *decimal places*. ViewStore won't allow more than this number of places to be typed in, and all numbers are displayed with this number of places. So if D is set to two, then 3.14159 can't be typed in; only 3.14 is possible. And 217 is displayed as 217.00. If D is left blank or set to zero, then no decimal places are used in any numbers; this is what to use for the 'Page' field of the bibliography.

Indexes

The names of any index files, and the length of their sort keys should be put in the record format, in the 'Key' and 'Index name' columns.

The file-names used must match those used when the SETUP utility was run; if they don't then an 'Index: filename not found' error message will occur. This can be sorted out by changing the name of the file the *RENAME COMMAND, or by altering the name in the record format. It is more convenient to have the filename the same as the field name (as with the Author field), but it can be any valid filename according to the filing system used. Of course, all the index files are in directory I, but the 'I' should not be used in the filename.

The key length should match that used for the SETUP utility too, but it doesn't have to; SETUP only uses the key length to work out the initial size of the index files. It is the key length in the record format that really counts, so make sure this is long enough to differentiate the data properly.

In the bibliography database, three index files were planned, one for the author, another for the title and also the date, but only the Author index was set up using the SETUP utility. The other two indexes can be built later, but their filenames and key lengths can be entered now. But if they are, then the 'I' field should be used to keep them switched off. This column can accept 'Y' for an active updateable index such as that for the Author field, 'R' for a read-only index, or 'N' for an inactive index or no index at all. If left blank, then no index is assumed. To switch off the non-existent Title and Date indexes, use N or a blank; use Y for the Author field.

Limits and Validation

The 'Low limit' and the 'High limit' are used to check each field entry as it is entered into the database. Obviously, values that are more than the lowest limit, and less than the high limit are acceptable, otherwise, 'Limit' is displayed in the status area of the screen and the entry is rejected. For example, to limit dates for articles in the database to those in a single decade, the limits '1/1/80' and '31/12/89' could be used. Make sure that the date follows the correct form, either dd.mm.yy, or mm.dd.yy, depending on which date system selected. Be careful, as it is possible to type in a limit like 1/1/1980 by mistake, and then all dates entered will be disallowed with a 'Bad date' error message.

This works quite simply for numeric type fields and dates, but is more complex with the alphanumeric and text fields. With alphanumeric fields, entries are compared by alphabetical order. A low limit of 'DAAA' means that 'David Cunningham' could be accepted into the final database, but 'Cunningham, David' wouldn't be because it is earlier in alphabetical order than the low limit. Similarly, a high limit of 'EZZZ' would allow 'Ernest Frobisher' but not 'Frobisher, Ernest'. No distinction is made between capitals and lower case, and if there is more than one word, it is the first entered that counts.

Text fields are different again. They act like alphanumeric fields as a rule, but if there is more than one word in the database entry, the limit check takes account of only the last word! This is not as it should be; the ViewStore manual suggests each word is separately tested against the limits, but in fact only the first or last is relevant.

Entries are not tested against any limits if the 'Low limit' or the 'High limit' fields are left blank.

The limit test gives fairly crude control over the data that can be typed into the database. Where it is vital that data falls into a particular format, or follows a certain coding system, two further things can be done.

First, a prompt can be given. The record format itself gives prompts: remember 'Type: A/T,N,D/M'. This is useful for two reasons. First, the prompt can give fuller details of the name or purpose of each field to be entered, particularly where the width set for the field is narrower than the field-name. Second, a list of permitted entries or codes can be given, or the format of the entry described. For example, the author field of the bibliography could be prompted 'Author of article: eg SMITH, Bruce'. The Prompt column lies off the initial screen, to the right of 'Index name'; pressing RIGHT scrolls the screen towards the 'Prompt' field, where the prompt text can be entered.

To limit entry to one of a few options, a complete list of the permitted entries can be put in the 'Value list'. If a coding system for the article references is worked out, ('AU' stands for Acorn User magazine, 'B' is Byte and so on), then the value list could contain the following for example 'AU,B,PCW'.

The complete format definition for the bibliography database shown in figure 9.3 uses many of the features discussed. When the format is complete, press the key for Command mode. This ensures that the format data entered is saved automatically in the format file F.BIBLIOG, which was created earlier by the SETUP utility.

```
L Space 29554
                          Record format
Field name

Field name......Wid.T.S.D.Low limit..High limit.I.Key.Index name.
AUTHOR          20 A Y                           Y   8 AUTHOR
TITLE           30 T Y                           N  10 TITLE
MAG              3 A N                           N
DATE             8 D N    1.1.70     31.12.99    N   3 DATE
PAGE             3 N N 0 1            300         N
KEYWORDS        30 T Y                           N
```

```
L Space 29554
                          Record format
Prompt

Prompt.............................................................
Author of article: eg. SMITH, Bruce
Title of article:
Magazine code: AU=Acorn User, B=Byte, PCW=Personal Computer World
Date of publication:
Page number:
Key words: eg. dump graphics printer FX80
```

```
L Space 29554
                          Record format
Value list

Value list.........................................................
AU, B, PCW
```

Figure 9.3. The completed format for the bibliography.

The Database Header

After the database format has been defined, there is one final set of data
to enter before the blank database can actually be used. This is the
database header. It contains general information about the database as
a whole, whereas the database format information all refers to specific
fields. Pressing SHIFT–f1 (DATABASE HEADER) gives access to the
database header: the blank header is shown in figure 9.4.

```
L Space 29554
                          Database header
Title

         Title
       Display
   Record size
      Capacity
   Index field
   Screen mode
```

Figure 9.4. A blank database header.

There are six pieces of information on the header. The 'Title' refers to the title of the database. If a title is specified, then it will appear centred in the status area of the screen when the finished database is displayed, in both Spreadsheet and Card display. Of course, a title isn't necessary, and it can be left blank.

The 'Display' line allows a choice of 'S' for Spreadsheet and 'C' for Card mode; whichever is chosen becomes the default display for the database. So if S is entered, then the Spreadsheet display will be used to show the database to begin with, but of course the Card display can still be used by pressing CHANGE DISPLAY (key f2). If 'Display' is left blank, then Spreadsheet mode is assumed.

Similarly, 'Screen mode' and 'Index field' control how the database is displayed when first loaded. For example, the screen mode might normally be set to '3', to show the maximum number of fields across the width of the screen. And the order in which the records are shown is controlled by the index field. So the bibliography database could be displayed in alphabetical order of author's name as soon as it is loaded in. Note that the name of the index field is needed and not the name of the index file for that field, though as in this case, they are often exactly the same.

The 'Capacity' line controls how many records ViewStore keeps in memory at once. How much spare memory there is depends on the screen mode, and the length of each record. The maximum is 50 records, but this is usually too many unless a second processor is used or the records are very small. A more usual number is 30. The maximum capacity is assumed if the database header is left blank. If the error message 'Record too big' occurs while using the database, it doesn't usually mean that one particular record is too big, but that the capacity is too high for the screen mode, so either change the mode or the capacity.

'Record size' is the most complex part of the database header. It controls the amount of space each record occupies on disc or in memory. There are two ways this can be used.

The simplest is to enter a number, for example '200'. This means that each new record can occupy up to 200 bytes. This gives space for a few less than 200 characters in the record. For the bibliography, that is probably enough: say 20 for the author's name, a maximum of 50 for the

title, a few for the magazine code and date, and so on. But this means that every record occupies 200 bytes of space, even if the actual data is only 50 bytes; imagine the waste of disc space if all the records were really short. Also, an especially long piece of data would have to be abbreviated; there is no quick way to extend the chosen length.

The alternative is to allow each record to be a different length: not all databases can do this, but ViewStore can and it makes more efficient use of space in memory and on disc. Just enter a record size of, say, '+35'. This means that however long the record starts off when it is typed in, an *extra* 35 bytes of space will be left. This should give enough space to alter a record, if a name was spelt wrong or an address had to be changed. However, this method is not always suitable. Imagine a name and address book list: some names and addresses are added 'whole' but some are added piecemeal; first the name and telephone number, then the address later. After adding a name, ViewStore leaves an extra 35 bytes, and that might not be enough for the address. All the other records can't just be moved up to make room. Using '+100' to give enough room for the address is possible, but that would add an extra 100 bytes on to the end of all the records, even if the address were typed in straight away. That might be more wasteful than using a plain '200' from the outset.

In general, the +35 method is best, but it depends on the circumstances. If the Record size is left blank, then +20 is assumed. In contrast to the address book, the bibliography database has all the information available right from the moment of typing in each reference, so only enough space for editing need be left, so +20 should be adequate.

The completed database header is shown in figure 9.5. Pressing DATA (key f0) returns ViewStore to the Spreadsheet display, and ESCAPE goes back to Command mode. This also ensures the database header information is saved as part of the format file, F.BIBLIOG.

```
L Space 29517
      Database header
Record size

       Title Computer bibliography
     Display S
 Record size +20
    Capacity 30
 Index field AUTHOR
 Screen mode 3
```

Figure 9.5. The completed bibliography database header.

Now the database is at the stage when it is ready to accept data. Some sample data is shown in figure 9.6; it's a selection of feature articles in the microcomputer press for early 1987. Try to either copy this information, or better still, get out a big pile of magazines and type in data relating to about 50 articles.

Figure 9.6 is displayed in alphabetical order of author. Remember to index the database *by entry* before adding any data:

```
f6
Fields: AUTHOR
RETURN
```

Notice as the data is entered that there is a pause as the first section of each record is typed in, whilst the disc driver whirrs. This occurs as ViewStore makes room for the new data and keeps the index of authors up to date.

```
L Space 31    Indexed by AUTHOR
                            Computer bibliography
Magazine: AU=Acorn User, B=Byte, PCW=Personal Computer World

AUTHOR...........TITLE...................MAG.DATE.....PAG.KEYWORDS..
KOELMANS, Albert   It's all Greek           AU  1.2.87     95 pythagoras
LAWRENCE, David    Variations on a Theme    AU  1.4.87     86 theme musi
LIARDET, Mike      Branching Out            PCW 1.5.87    166 Prolog Al
LIARDET, Mike      Balancing the Scales     PCW 1.4.87    152 Prolog Al
MALCOLM, Peter     Caught in the Net        PCW 1.5.87    134 network mu
MEGARRY,Jacquetta  Wisdom with Words        AU  1.6.87    108 wordproces
NEWMAN, Robin      The Network Page         AU  1.2.87    173 Econet net
NICHOLSON, Rod     Explain That to Me       AU  1.3.87     90 Eliza lang
PELLI, Denis       Programming in PostScrip B  1.5.87    185 dtp postsc
QUINN, Tony        Editing Your Own         AU  1.6.87     87 magazine d
SCHIFREEN, Robert  Lotus Manuscript         PCW 1.4.87    132 wordproces
SEYBOLD, John      The Desk Top Publishing  B  1.5.87    149 dtp
SICHERER-ROETMAN, A Stitch in Time          AU  1.2.87     88 cross-stit
```

Figure 9.6. The bibliography database in use.

Laying Out the Cards

ViewStore's Spreadsheet mode can look very unfriendly: there can be too much information displayed at once. Take for example figure 9.6. The screen is crowded with information, yet when the database is in use, probably only one of the rows at a time will be of interest. It isn't quite so easy to browse through the database.

Depending on the application, the Card display can be better. Compare figure 9.6 with figure 9.7. Although there are only two of the references on the screen at once in Card mode, the display works in exactly the

same way as the Spreadsheet mode: DOWN switches to the next record, RIGHT goes to the next field, SHIFT–LEFT returns to the first field of the current record, and so on.

```
L Space 31    Indexed by AUTHOR
              Computer bibliography
Magazine: AU=Acorn User, B=Byte, PCW=Personal Computer World


    AUTHOR KOELMANS, Albert                    MAG AU
     TITLE It's all Greek                      DATE 1.2.87
  KEYWORDS pythagoras recursion trees gra      PAGE 95

    AUTHOR LAWRENCE, David                      MAG AU
     TITLE Variations on a Theme               DATE 1.4.87
  KEYWORDS theme music autumn tunemaker        PAGE 86
```

Figure 9.7. The bibliography in Card display mode.

The Card mode can be used while the data is being entered, or while browsing through the database. In all respects, it is interchangeable with the Spreadsheet display.

Card Display

Press f2 (the CHANGE DISPLAY function key) to switch to the Card mode. If there is any data in the database, then the initial Card screen is hardly less daunting than the Spreadsheet mode. But this can be remedied. Press CARD LAYOUT (key SHIFT–f0) to change to the *screen designer*. The screen designer can't be entered direct from Spreadsheet mode.

The initial card design shown in figure 9.8 is not ideal. All the fields are shown in the order they are in the record format, with the space reserved for each (that is the field width) shown as an underline. The underlines and fieldnames simply flow over from one row to the next, with no consistent formatting of the layout.

```
                        CARD LAYOUT
AUTHOR _____ TITLE _____ MAG ____ DATE __
_____ PAGE ____ KEYWORDS _____
```

Figure 9.8. The initial Card layout.

This screen can be edited to improve the layout of the card. A tiny cursor flashes under the first field; it can be moved about with the cursor keys. As the cursor moves, it 'wraps around' from one edge of the screen to the other; take it off to the left and it reappears to the right. It also

wraps around top to bottom. Position the flashing cursor over one of
the fields, either the name part or the underline part. Now pressing
COPY 'picks up' that field: this is marked by the first character of the field
name changing to a '*'. Once picked up in this way, the field can be
moved to some other place on the screen by moving the cursor to the
place where the field should go, and then pressing COPY again. The first
letter of the fieldname is moved to where the cursor is.

It doesn't matter if the field destination overlaps the original place; even
very small adjustments can be made this way. But the destination
mustn't overlap any of the other fields, otherwise the field is put
somewhere else. In fact it's put in the first place further down the screen
where it will fit. Note also that like the cursor, the fields 'wrap around'
if there is too much underline to fit at the right of the screen, it runs off
the right edge and reappears at the left of the screen a row lower down.

If you have trouble with this, then the best strategy is to move all the
fields down to the bottom of the screen, to get them out of the way.
Then design the screen by picking out one field at a time from the bunch
at the bottom, and placing each where it should go.

Try to arrange the fields like this:

```
   AUTHOR  _____
    TITLE  _____
      MAG  __
     DATE  _____
     PAGE  __
 KEYWORDS  _____
```

with the Author field on the first row of the designer screen. The top of
the area isn't marked, so check it is the first row by moving the flashing
cursor to the Author row, then moving it up a single line; it should wrap
around to the bottom of the screen if it is the first row.

Now switch back to the Card display by pressing DATA (the f0 function
key). If the card layout is like that just mentioned, with six rows of data
starting from the very top of the designer screen, then three cards of
information are shown on the card display screen, with a single line of
space between them.

If more space is required to make the break between cards more
obvious, then this can be added. Return to the designer with SHIFT–f0,
then move each of the fields down two lines. Move the last field first, or
there will be trouble with overlapping! This places the Author field on

128

the third row of the designer screen. The cursor should take three upward moves before it switches to the bottom of the screen. Now go back to the Card display (press f0) which should show only two cards. The extra space introduced at the top of each card separates the two much more clearly than the single blank row.

The screen designer is probably the weakest section of ViewStore in comparison to other databases. Many other packages let you 'paint' cards. Certain fields can be highlighted and text annotations or boxes can often be added to the card. This can't be done in ViewStore. The only thing that can be changed is the layout of the fields across the card. Figure 9.9 shows the final design of the bibliography card, with four blank rows at the top of the card.

`CARD LAYOUT`

```
      AUTHOR _____        MAG __
       TITLE _____        DATE _____
    KEYWORDS _____        PAGE __
```

Figure 9.9. The final bibliography Card design.

The only improvement that can be made in some cases is to add one or two *dummy fields* to the Record format. These might have field-names like '----------', and a width of zero. With a zero width, no data can be added to these fields, and they don't appear on the Spreadsheet display at all. But the fieldnames do show up on the Card display, and using the screen designer they can be positioned adjacent to the most important fields, to 'underline' them:

```
    AUTHOR _____  MAG ___
           --------------
```

On the Card display itself, this appears as:

```
    AUTHOR Graham Bell    MAG AU
```

The limitation with this is that the length of a fieldname can't exceed 15 characters, so the 'underline' can only be that long.

Screen Modes and Colours

Owing to the differences in screen width and depth between modes, each database format is meant for a particular mode. For example, a Card display designed for mode 3 won't fit on a mode 6 screen. This is why the mode specification is in the database header.

Whatever mode is used (except mode 7), the screen colours can be set up by the method used within ViewSheet, which is fully described in Chapter Five. For example, to select a dark blue background in ViewStore Command mode, hold down CTRL, and type:

```
S04000
```

This colour will be maintained until another colour is set or the screen mode is changed. Pressing ESCAPE to look at the data doesn't change the colours.

Sometimes a 'hidden' mode change can take place. For example, mode 6 should be selected before running a utility program, to make sure there is enough spare memory. If the colours are set up in mode 6 and ESCAPE pressed to return to the Data display, the original screen mode is selected automatically, and as a consequence the colours are lost.

If no screen mode is specified in the database header, then there can be other problems. No mode changes happen automatically when entering the Data display, so the colours aren't lost. But if the Card layout doesn't fit the size of screen, then the whole thing reverts back to it's original, unformatted look, as shown in figure 9.8. For this reason, always set the mode number in the database header.

Creating New Indexes

When starting up the Bibliography database, three index fields were discussed, but in order to speed up data entry, two of them were not used. When the majority of the information has been entered, the two new indexes can be created.

The INDEX utility program is designed for this. But before running it, the name of the index and its keysize must be put into the Record format, if they are not already there. Also, the new index must be marked as active by putting either 'R' or 'Y' in the adjacent 'I' column.

Y marks the index as active and *updateable*. That means that once the index has been created, ViewStore will keep it up-to-date automatically. In all respects, the new index will be similar to the Author index which was originally built up as information was typed into the database.

R marks the index as *read-only*. That means that ViewStore doesn't try to keep the index up-to-date. If elements within the data are modified, for example a spelling, then the index is not changed. This means that the modified data will appear out of correct alphabetical order. Figure 9.10 shows the two new index filenames added to the Record format, along with their Y and R index types.

```
L Space 27513
                          Record format
Field name

Field name......Wid.T.S.D.Low limit..High limit.I.Key.Index name..
AUTHOR          20 A Y                             Y   8 AUTHOR
TITLE           30 T Y                             R   8 TITLE
MAG              3 A N                             N
DATE             8 D N   1.1.70     31.12.99       Y   3 DATE
PAGE             3 N N 0 1          300            N
KEYWORDS        30 T Y                             N
```

Figure 9.10. Amending the Record format to create new indexes.

To create the index files themselves, the utility INDEX should be run. Return to ViewStore Command mode and type:

```
UTILITY INDEX
```

The utility first asks whether to use a *select file*. This allows creation of an index of just some of the records in the database, but the selection hasn't been covered and in this case a complete index is desired.

Then the fieldname needing to be indexed is prompted for. This is not the name of the file, though they are often similar. The utility program then constructs the index file. The whole process looks like this:

```
=>UTILITY INDEX
INDEX
Use select file (N,Y)? N
Field name? TITLE
I.TITLE
Building index.........................................
39 records processed
=>
```

The Date index can be built in the same way. As soon as they have been built, the new Title and Date indexes can be used to sort the records in the database. In Data mode they appear in the list of fields available on pressing INDEX FIELD:

```
f6
Fields: AUTHOR, TITLE, DATE
TITLE
```

This sorts the records into alphabetical order of the article title.

Remember that because the relevant index can't be updated, this Title order is correct only until one of the titles is modified (within the appropriate key length). This also applies if a record is deleted, or if new data is added to the end of the database. In this case, the read-only indexes should be rebuilt. If new records added without rebuilding the Title index, then they won't show on screen when the records are put in title order. Only the old records that the index file 'knows about' are shown. In contrast, the updateable Date index need never be rebuilt.

10: Searching, Selecting, Sorting
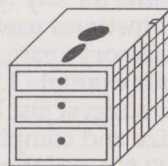
The Bibliography Database

There is a fundamental difference between a database and an archive. The latter is a collection of dead material that's often difficult to sort through, whereas a database is a living store of information, easily accessible and always growing as new facts are gathered and added.

Browsing through the database, sorting it into a particular order and selecting one of the indexes, then searching through the lists of information for items of interest is not the best way of finding what you want. The computer is good at repetitive searching, so get it to do the work for you!

ViewStore has a series of utility programs that allow easy access to database information in a variety of ways. One of these, the LABEL utility was described in Chapter Seven.

Simple Selections

The most fundamental utility program is SELECT. This utility can choose a small set of the records and display them on screen or to be printed out. This utility is doubly useful because it can produce a sub-list of records that can be used with another utility program. For example, a selection of name and address records can be used with the LABEL utility to produce labels for just some of the people on a database.

It is important to remember that the records chosen by SELECT must have a common linking factor- be it simple or complex. Thinking of the Bibliography database, a simple link could be all the articles written about printers; a more complex one could be all the articles on printers or graphics that appeared in Acorn User magazine between March and May 1987, that were not written by John Knight.

So how does it work? First, load the database. As an example, load the Bibliography database from Chapter Nine (the procedure is broadly the

VAV—J

same for any type of database). Before starting SELECT or any utility, it's sometimes useful to change to a screen mode which leaves adequate memory spare, say Mode 6 on a BBC model B, or into a shadow mode (for example 131) on a model B+ or a Master series micro. It's difficult to suggest just how much memory is required, as this will depend on the size and complexity of the records in the database. As a guide, the SELECT utility requires about 2500 bytes free for the actual program, plus extra bytes for the format file and some of the data. Another good idea is to type in:

```
LIST
```

This shows on screen a list of the field names used in the database; the names can be useful to jog your memory. Figure 10.1 shows the result for the Bibliography database, with the fields named and numbered.

```
1   AUTHOR              2   TITLE
3   MAG                 4   DATE
5   PAGE                6   KEYWORDS
```

Figure 10.1. List of the Bibliography fieldnames.

Still in ViewStore Command mode, type:

```
UTILITY  SELECT
```

to run the utility program.

If ViewStore responds by displaying the message 'Insert utility disc & hit a key on screen', then it means that for one reason or another ViewStore can't find the utility. First, SELECT may be spelt wrongly: if so, then press ESCAPE and start again. Second, the disc containing the utility programs may not be in the drive, in which case simply insert it and hit the space bar. Third, it may mean that the prefixes have been set up wrongly, or the utility disc is in a different drive to that suggested by the prefixes. If so, press ESCAPE and set up the prefixes properly.

Once the SELECT utility is running, it displays the following on screen:

```
List or create select file (L,F)?
```

To create a subset of the database which can be used later with another utility, press F. We will cover this in more detail later. To just list all the articles on printers, press L and RETURN. Simply pressing RETURN has

exactly the same effect, because L is the default choice here. Next, SELECT asks whether to print on the screen or on a printer:

```
Screen or Printer (S,P)?
```

As shown by the brackets, the screen is the default choice. Choosing the printer option without a printer connected will cause the machine to seize up; both the CAPS LOCK and SHIFT LOCK lights on the keyboard will light up. If this happens just press ESCAPE and CTRL-C.

Next the utility asks for the *select criteria*. This is the common factor linking all the records you want to select together. To find all the articles relating to printers from the Bibliography database, using the Title field would be misleading, as not all the required articles would have the word 'printer' in their title. The Keyword field is more useful; the choice could consist of all records having the keyword 'printer':

```
Select criteria? KEYWORDS = PRINTER
```

This means select all the records where the Keyword field contains the word 'printer'. After RETURN has been pressed, ViewStore repeats the 'Select criteria' prompt; just press RETURN on its own to indicate that you have finished your selections. The utility then sifts through the database, and shows on screen the records which match the selection criteria. With the example bibliography database, this produces four records. Figure 10.2 shows the whole process, and lists the four matching records.

```
=>UTILITY  SELECT
SELECT
List or create select file (L,F)? L
Screen or Printer (S,P)? S
Select criteria? KEYWORDS = PRINTER
Select criteria?


KNIGHT, John      Doing it on the Side    AU   1.4.87    96
YETTS, George     Inspired Inscriptions   AU   1.5.87    96
WILLIAMSON,Clive  Soft Options            AU   1.6.87   102
SCHIFREEN,Robert  Lotus Manuscript        PCW  1.4.87   132

4 records selected out of 39
=>
```

Figure 10.2. Selecting articles on printers.

These records aren't listed in any particular order, even though according to the database header the database should be indexed by author as soon as it is loaded. In fact, the selections are always listed in the order they are stored in the main database file. In a similar way, if in Data mode and no specific index file is used, then 'Indexed by entry' is shown in the status area of the screen.

The records are shown much as they would appear in Spreadsheet mode, with as many fields as possible arranged across the screen.

More Complex Selections

Let's approach the more complicated selection mentioned earlier— all the articles on printers or graphics that appeared in Acorn User between March and May 1987 that weren't written by John Knight.

Selection criteria can be combined, as is shown in figure 10.3. This figure illustrates a variety of points. First, = (equals) is not the only comparison that can used. The following can also be used:

>=	greater than or equal to	>	greater than
<=	less than or equal to	<	less than
<>	not equal to		

Applied to dates, < and >= clearly mean 'before' and 'on or after'. In text and alphanumeric fields, the data is compared alphabetically.

```
=>UTILITY SELECT
SELECT
List or create select  le (L,F)? L
Screen or Printer (S,P)? S
Select criteria? DATE >= 1.3.87
Select criteria? AND
Select criteria? date < 1.6.87
Select criteria? and
Select criteria? KEYWORDS = printer
Select criteria?

KNIGHT, John    Doing it on the Side    AU  1.4.87   96
YETTS, George   Inspired Inscriptions   AU  1.5.87   96
SCHIFREEN,Robert Lotus Manuscript       PCW 1.4.87  132

3 records selected out of 39
=>
```

Figure 10.3. Using multiple selection criteria.

Second, individual selection criteria can be strung together with AND, so that all must be met. The third record that was selected in figure 10.2 has a Date field containing '1/6/87'. This matches the first test in figure 10.3, but fails the next and so fails overall. OR can be used as well as AND, to link together criteria where the record should be selected if either option is true.

It doesn't matter if you use capital letters or lower case ones—AND is the same as and, date is the same as DATE, and printer is the same as PRINTER and so on. So upper or lower case is ignored in fieldnames, in AND, in OR and in the data itself.

As before, RETURN on its own ends the list of selection criteria.

Extending the Criteria

Figure 10.4 shows all the instructions to type in to select all the articles on printers or graphics that appeared in Acorn User between March and May 1987 which were not written by John Knight.

When selections are this complicated, then it is probably worth writing them all down before using the SELECT utility. This is for two reasons. The most obvious is so that you don't forget part of the selection. Less obvious is the fact that AND and OR don't have any precedence - SELECT just deals with them in the order they are typed in. This means that for certain choices, brackets are necessary to make the correct selection. Brackets should be used to make your choices crystal clear, even when they might be unnecessary - they can't do any harm.

Think about the difference between:

```
AGE >= 50 AND AGE < 60 OR AGE >= 30 AND AGE < 40
```

and the following:

```
(AGE >= 50 AND AGE < 60) OR (AGE >= 30 AND AGE < 40)
```

Pick a few ages and work through the first version, one part at a time, left to right. This expression can't select people in their 50s, because the final criterion is AND AGE < 40. The second version correctly brackets the criteria so that people either in their 30s or their 50s are selected.

```
=>UTILITY SELECT
SELECT
List or create select file (L,F)? L
Screen or Printer (S,P)? S
Select criteria? DATE >= 1.3.87
Select criteria? AND DATE < 1.6.87
Select criteria? AND (KEYWORDS = PRINTER
Select criteria? OR KEYWORDS = GRAPHICS)
Select criteria? AND MAG = AU
Select criteria? AND AUTHOR <> "Knight, John"
Select criteria?

JOHNSON-DAVIES,D Return of the Mandelbr AU 1.5.87  81
YETTS, George    Inspired Inscriptions  AU 1.5.87  96

2 records selected out of 39
=>
```

Figure 10.4. The final complex selection.

Notice the placement of AND and OR in figure 10.4. They are all at the beginning of a line. This is convenient and conventional, but in fact there is no difference between:

```
AGE >= 30
AND AGE < 40
```

and the following:

```
AGE >= 30 AND
AGE < 40
```

or even:

```
AGE >= 30
AND
AGE < 40
```

Note that you can't split an individual comparison as follows:

```
AGE >=
30
```

It is worth noting the quotes around "Knight, John" in the final selection criterion. Quotes should always be used around fieldnames or data that

contain spaces, punctuation or other special characters. Only ordinary numbers, dates and single words can be left without quotes.

You can use single or double quotes. Other characters can also be used, but they are not recommended. Only if the data itself contains both single and double quotes should the hash (#) or exclamation mark (!) be used, for example.

Any one selection line can contain just a single condition, or a combination of several criteria. The line can be up to 255 characters long. As all the conditions are stored in the micro's memory, it is possible to run out of memory if the selection is too complicated. This shouldn't happen if the screen mode is changed to mode 6 or a shadow mode before you start the selection process.

Alphanumeric and Text Fields

By looking at the results of selections, the difference between an alphanumeric and a text field can be seen. Take a look at this record:

```
Author.....    Title..............   Mag Date.. Page Keywords.....................
KNIGHT,  John  Doing it on the Side AU 1.4.87 96   spreadsheets sideways printer
```

The important point to notice is that 'Keywords = printer' selects this record, because the Keywords field is a text field, and so the pattern 'printer' is compared with the data *a word at a time*. Conversely 'Author = John' doesn't work, because the Author field is alphanumeric - the data is always treated as a whole. So text fields are selected if they *contain* the pattern word, whereas alphanumeric fields must be *exactly the same* as the pattern word. Even 'Author = Knight   John' does not work, because there are two spaces, but only one in the data.

In all other respects, alphanumeric and text fields are the same and because of their extra flexibility in selection, text fields are better to use.

Printing Selection Results

Before printing any results from ViewStore, it may be an advantage to load a printer driver. This isn't strictly necessary for simple results, but becomes so if any complex printer effects are required.

ViewStore uses exactly the same printer drivers as VIEW and ViewSheet, so if the appropriate driver for the printer is available, then it can be loaded with a command similar to:

```
PRINTER FLO16
```

from ViewStore Command mode. This loads the correct FLO16 driver for the Ricoh Flowriter range of daisy-wheel printers.

ADFS, Net: A fuller filename may be required if the printer driver is in another directory, for example:

```
PRINTER &.LIBRARY.FLO16
```

The printer driver allows all characters to be printed as they appear on screen, including the pound, dollar and hash signs.

Details of printer drivers for the View family are outlined in Chapter Six. It may also be necessary to select the network printer (with *FX 5 4) or a serial printer and the appropriate baud rate (with *FX 5 1 and *FX 7 and *FX 8). The details for this are given in Chapter Two.

Whether or not a printer driver is loaded, when the printer is set up, a printed record of the results of a selection can be obtained by answering 'P' to the question:

```
Screen or Printer (S,P)?
```

The SELECT utility then prompts:

```
Printer width (80)?
```

80 is the default printer width, so just press RETURN if the printer is 80 characters wide. A wider printer is capable of showing more fields across a row, so type in the character width that printer is capable of. A common width for wide-carriage printers is 132 characters.

Other SELECT utility prompts follow the same pattern as before. When the criteria for selection are completed, then the records selected are printed out, just as they are shown on screen.

Preparing a Select File

Using the SELECT utility in this way is a simple introduction to its real function; it's really meant for creating *selection files*. A selection file is a file on disc that contains a list of the records that match particular conditions. So far that's just like the selections above. But the list of records in a selection file can be sorted into order and then used by other utilities. So for example a list of people could be selected from a database, and labels prepared for each of those selected. The list could be sorted alphabetically by name, according to the age of the addressee, or according to any other information in the database.

The records can't be sorted unless a selection file has been created. Records can be listed in unsorted order only - that is in the order they were added to the database.

To create a selection file, answer 'F' at the initial prompt:

        List or create select file (L,F)? **F**

Next the utility prompts for the selection conditions in exactly the same way as before. A blank line signals the end of the conditions, and the utility then asks according to which field the records should be sorted:

        Sort field?

Type in the name of a field, for example 'Author' followed by RETURN.

Some lists are sorted according to more than one field. Think of the telephone directory. There are thousands of Smiths, they can't be distinguished by their last names, so they are arranged by initials and first names. So Smythe, B preceeds Smythe, C, but both follow Smith, Z. The surname is the primary sort field, the forename is a secondary sort field. The records are sorted according to the primary field first and where two can't be distinguished, their secondary fields are compared. If they are still identical after a secondary sort, a third field can be compared too, and so on.

A ViewStore selection file can be ordered according to several secondary fields by typing in the fieldnames, one at a time, at the 'Sort field' prompt. If you don't need it sorted by a secondary sort field, just press RETURN. Similarly, pressing RETURN alone terminates the list of secondary sort fields.

The final part of a selection and sort is to tell SELECT whether to sort the records into ascending or descending order. Ascending results in conventional alphabetical order, or date order for date fields, or number order for numeric fields. Descending is the opposite, of course!

The whole process of creating a selection file containing the Bibliography records, which refer to articles on printers, is shown in figure 10.5 below.

```
=>UTILITY SELECT
SELECT
List or create select file (L,F)? F
Select criteria? KEY* = PRINTER
Select criteria?
Sort field? AUTHOR
Sort field? DATE
Sort field?
Key width: 11
Ascend or Descend (A,D)? A
Selecting...........................
4 records selected out of 39
SORT
Sorting...
=>
```

Figure 10.5. Creating a sorted selection file.

Wildcards and Field Numbers

One other new element shown in figure 10.5 is the use of a wildcard. 'Key*' is used as an abbreviation of the full field name 'Keywords'. To save typing the whole fieldname out, * can be used to match any number of letters, or ? can match any single letter. Wildcards can also be used when specifying values in selection conditions. So:

```
AUTHOR = SM*
```

could be used to select Smith, Smythe and Smallbone. The '*' matches any number of letters after the 'Sm'. Note that this includes no letters too, so an author called just Sm would also be selected.

The field number can also be used in place of a full fieldname. This is the number listed by the fieldname when the LIST command is used. The Author field is field one, the Date field is number four, as is shown in figure 10.1. So the situation could be expressed as:

```
1 = SM*
```

which gives exactly the same results.

In the unlikely event that a fieldname is a number, then the field called '5' is used, rather than field number five.

Specifying Longer Key Widths

In figure 10.5 you can see the line:

```
Key wifth: 11
```

This means that ViewStore is taking 11 characters into account when sorting the records into the correct order. How does it know to use this number of characters to distinguish the records from one another?

Basically, when a sort field such as the Author field is used, the SELECT utility always tries to use the key width specified in the Key part of the record format for that field. For this reason, it can be useful to put in a key width even if no index is planned for that field. So the 11 characters of the key in figure 10.5 are made up of eight for the Author field and three for the Date field, as specified in the record format.

However, there are four exceptions to this rule. First, if no key width is given in the record format, then the display width in the wid part of the record format is used. Second, if this is also blank, then the default display width of 18 characters is used. Third, date fields are always sorted with a key width of three, whatever their stated key width or display width is.

The fourth exception is when a key width is specified explicitly along with the sort field. In the Bibliography database, the Author field has a key width of eight set in the record format. This controls the key width

143

used when creating the index file for that field. Naturally when a sort is called for, the same key width is used. This can be overridden, and a key width of, say, 10 can then be used for the sorting like this:

```
Sort field? AUTHOR, 10
```

If this were done, then the combined key width for Author and Date fields would be 13 characters.

This is only worth doing if the key width specified in the record header is too short to completely differentiate the records. Too long a key just slows the sorting process down. The combined sort key length cannot exceed 250.

Sorting Suggestions

Sometimes no selection is actually needed, but the records must still be sorted into order. For example, you might want to prepare labels for all the people on a database, and it's convenient to have the labels printed in alphabetical order.

In the utility directory, along with the files U.LABEL and U.SELECT, there is also U.SORT. However, this can't be used on its own. There is no UTILITY SORT command. This file can be used only by the SELECT utility.

So to create a sorted file containing all the records, just press RETURN in response to the first selection prompt, as shown in figure 10.6.

```
=> UTILITY SELECT
SELECT
List or create select file (L,F)? F
Select criteria?
Sort field? NAME, 11
Sort field?
Key width: 11
Ascend or Descend (A,D)? A
Selecting.............................
51 records selected out of 51
SORT
Sorting......
=>
```

Figure 10.6. Creating a sorted file containing all records.

It is worth noting that the selected and sorted records from a database called BIBLIOG go into a selection file called S.BIBLIOG. There is a separate prefix used for these selection files, which can be set up with the following command:

```
PREFIX S  :2.
```

If this prefix is set, then the SELECT utility creates a selection file called ':2.S.BIBLIOG'.

The name of the select file created is fixed. If a second selection is made, then the new selection file overwrites the old one. All trace of the old file is lost. With a very large database which doesn't change rapidly, it's convenient to keep all the resulting selection files rather than overwriting them, because it can take a long time to recreate them if the same selection is needed in the future. To do this, all you need to do is rename the selection file before you create a new one. For example, like this:

```
*RENAME  :2.S.BIBLIOG  :2.S.BIBLIO4
```

However, the new S.BIBLIO4 file will have to be renamed back to S.BIBLIOG before it can be used again.

If the database is very large, then the SELECT utility may create a temporary file called S.SRTINT, plus whatever prefix is in use. This file is automatically deleted when it is no longer required, and it should never appear in a catalogue of the disc. But it will destroy any other file called S.SRTINT.

Selection Files and Utilities

After creating a file containing the desired selection, the list can be used with all of the other ViewStore utiities, so that the utility is applied to only some of the records in the database.

Chapter Seven described the use of the LABEL utility to print address labels for the companies listed on a database called CREDIT, which is supplied on the programs disc accompanying this book. A selection file can be created containing only some of the records from the database using the SELECT utility. For example, a selection of the records for which:

```
ADDR*2 = DERBY
OR ADDR*3 = DERBY
```

should produce a group of just six records pertaining to the fictional companies based in Derby. Note the use of wildcard characters within the fieldnames.

This selection allows labels to be printed for only those six records. When the selection file is complete, the procedure for printing the labels is the same as that in Chapter Seven, except that the prompt:

```
Use select file (N,Y)?
```

should be answered with a 'Y'.

The first thousand copies of the ViewStore utilities disc contained a bug in the LABEL utility, so it couldn't be used with a selection file. The 'Use select file' prompt was never printed. An upgrade for this disc is available from Acorn.

Sorted Lists

The major problem with the SELECT utility is that either the selected records are printed out, or they are sorted, but not both. A sorted list can't be displayed.

The way around this is to use the SELECT utility to select and sort the records, and another utility to display the information. REPORT is a complex utility, but the complicated bits can be left until later. In its

simplest form, it can display or print the sorted list of records from a selection file.

When the selection file has been created with sorted records, (as shown in figure 10.5), you can run the REPORT utility. The utility normally asks only three questions. First, whether to use the select file, and second whether to use the printer or display the records on the screen. The third prompt asks whether to use a *report format file*, here the answer should be 'N'.

The utility then displays as much of each record as it can fit across the screen or paper, this is shown in figure 10.7. Notice the similarity with figure 10.2, although the fieldnames are printed at the head of the list of records.

```
=> UTILITY REPORT
REPORT
Use select file (N,Y)? Y
Screen or Printer (S,P)? S
Use report format file (N,Y)? N

AUTHOR                TITLE                 MAG DATE   PAG
KNIGHT, John          Doing it on the Side  AU  1.4.87  96
SCHIFREEN,Robert      Lotus Manuscript      PCW 1.4.87 132
WILLIAMSON,Clive      Soft Options          AU  1.6.87 102
YETTS, George         Inspired Inscriptions AU  1.5.87  96
=>
```

Figure 10.7. Displaying sorted records with REPORT.

To print the list out on paper, additional questions are asked about the printer. The maximum number of characters printed across the paper can be specified, with a default of 80. The page length can also be controlled.

If the printer uses continuous stationery, then the correct number of lines per page should be typed in. The default is 66 lines per page, which is suitable for most printers and normal listing paper. Seventy is more normal for continuous A4-paper, that is 11⅔ inches at six lines per inch. Therefore to the:

```
Single sheets (N,Y)?
```

prompt, answer 'N'.

147

When a single sheet of paper is positioned correctly in the printer, perhaps an inch of paper sticks up above the print head, reaching up to the adjustable rollers. If single sheets are being used, then reduce the true page length by about six or eight lines and enter perhaps 62 for A4–paper. This takes account of the unused paper above the print head. Then answer 'Y' to the single sheets prompt. This makes REPORT pause at the beginning of every page, so a new sheet of paper can be inserted.

The utility presents a prompt:

```
Page..
```

When the paper is set up, press the space bar to print the page, or press 'M' to miss out that page. On each page, REPORT prints the field titles and a number of records, one per line. There are four less records per page than there are lines per page.

Sense and Sensibility

As a database grows, the information in it becomes more and more valuable. The flexibility of an electronic database means that data is accessible in a variety of ways and many applications impossible with traditional paper records may be possible. But the data also becomes more and more costly to replace. The time and effort involved in collecting and collating information for even a medium-sized database is immense.

It makes very good sense to keep back-up copies of your database, preferably in different locations. For the vast majority of databases, where their contents don't vary rapidly, then a weekly or half-weekly back-up session should be sufficient. The best strategy is to keep, say, two sets of back-up discs, and copy the whole database on to the least recent set of discs at each session, using the *BACKUP command. So at any time there will be the original database, one back-up less than a week old and another less than a fortnight old. At the next session, the database should be backed-up onto the discs that are a fortnight out of date.

If a disaster does happen, the most recent back-up discs can be used to restore the database to its original condition without too much extra work or worry.

Net: Consult the network manager about making back-ups of the database, and about the procedures for restoring a damaged database.

It is very important to bear the Data Protection Act in mind. An information pack about the Act containing a booklet and application forms is available from Post Offices, and further information can be obtained from the Data Protection Registrar, Springfield House, Water Lane, Wilmslow, Cheshire, SK9 5AX.

If no personal information relating about people is kept, then registration is unnecessary. If information concerning living people is kept, then registration may be necessary. A very simple mailing list or record of accounts is unlikely to require registration. But it is still wise to read the booklet.

If any personal information database is kept, even on behalf of a club or other group, then it is a sensible precaution to make sure that all the people on the database know their details are included. Each person should have the chance to inspect their own entry, and have it amended or removed. Clubs could perhaps make it a condition of membership that electronic records be kept. This is the most difficult area for the Act, but providing no sophisticated use is made of the information and providing it is kept private, then probably there is no need to register.

Registered companies that keep data on employees, customers, suppliers or any other personal data *excluding simple payroll records* should register. Even a payroll that includes employees age or sex, for example, is not a simple payroll, so it's almost certain that companies using this type of information should register. Schools should usually be registered through their LEAs.

11 : A Database at Work
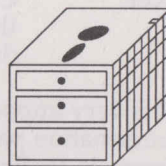
The Name and Address Database

The commonest form of database - the name and address list - is also one of the simplest in structure. Everyone keeps an address book, clubs keep membership lists, companies keep mailing lists, schools keep pupil rolls. And this name and address list is an ideal application for explaining and using ViewStore.

Planning

Before setting up the name and address database, it's best to decide what data needs to be kept, and in what way you want to store it and how to access it. Clearly, the names of the people involved, and their addresses are the most important items, but for, say, a sports club, the types of sport they each play may be important too.

Taking a club membership list as an example, some of the typical fields for the database are listed in figure 11.1.

```
Title           Mr, Mrs, Ms, Dr and so on
First name      or initial with full stop
Surname         the only indexed field?
Street 1        plenty of lines for even the
Street 2        longest addresses
Street 3
Town
County
Postcode
Phone number

Membership      full or just social member?
Number
Paid            dues paid?
Date next due   paid up until
Region          part of country
Interests       which areas of interest
Notes           extra room for other items
```

Figure 11.1. Fields for projected name and address database.

The first three fields are fairly standard, but although the list of names
will usually be kept alphabetically, the Surname field is not listed first.
Imagine what the spreadsheet display would look like:

```
Mrs Bridget Cartwright
```

this is much easier on the eye than:

```
Cartwright Bridget Mrs
```

The point to note with first names is that members whose first name is
not known must be given an initial like 'B.'. This is important, because
with a mailing list, an initial can be used on the envelope label, but
would look very odd if it were used in the letter itself:' Dear Bridget' is
fine, but ' Dear B.' can't be accepted. So it is vital that people with just
an initial can be selected from the database and sent a subtly different
letter, perhaps saying 'Dear Mrs Cartwright'. Now it's simple enough
to use the SELECT utility to find those records where the the first name
field is a single character. Use the wildcard that matches any single
character, '?', to select all those records where:

```
First name = ?
```

But what about Dr E F Connolly? Some people insist on using two
initials, and their records would not be selected. So it is best to record
them all as follows:

```
Mrs B.    Cartwright
Dr  E. F. Connolly
```

These can both be selected by pulling out all those records with a dot as
the second character. Remember the '*' wildcard can match anything,
even zero characters, so this can be done with:

```
First name = ?.*
```

Another way of selecting the records' Mrs B Cartwright' and 'Dr E F
Connolly' would be to use:

```
First name = ?   OR
First name = "? ?"
```

Sometimes it is even necessary to have an extra field in a mailing list database, especially for selecting which people should receive a formal or informal greeting at the beginning or end of a letter. This might be of particular importance in many of the professions, where some clients are merely acquaintances, but some are also personal friends.

The next group of fields is concerned with the member's address. The only important point is to not assume that all addresses are the same length. Leave separate fields for the town, the county and the postcode so that selections can be made. Very short addresses will have the Street 2 and Street 3 fields blank, but this doesn't matter.

The final group of fields gives the member's details. The membership number, unique to each member, and whether this year's fees have been paid, are important in most clubs. Less obvious are lists of special interests; an art association may want to select a list of members who are interested in particular forms of art. Sports clubs often have different categories of members; social membership, squash only, or full membership for example.

The region field gives a chance for a list of members from particular parts of the country to be drawn up. This could be done by address, picking all those who live in a particular county, but a wider grouping might be advantageous. Perhaps categories like Scotland, the North East, the Midlands and so on.

Setting Up

The main part of the format for the database is shown in figure 11.2. This closely mirrors the initial ideas shown in figure 11.1.

It is always a good idea, when there is a limited number of choices for a field, to list all the possible values in the Value list field. The lists of permissible values for the Title, Membership, Paid and Region fields are shown after the prompts too. This reduces the number of errors in the same way as limiting date or numeric entries with the low and high limits. Putting the permissable values in the prompt serves as a reminder when data is being entered.

L Space 28883
Field name

```
Field name......Wid.T.S.D.Low limit..High limit.I.Key.Index name
Title            4 A N                              N
First name      10 A Y                              N
Surname         10 A Y                              Y 10 NAME
Membership       1 A N                              N
Number           4 N N 0 0             9999         N
Paid             1 A N                              N
Date next due    8 D Y     1.1.70      31.12.99     N
Street 1        20 A Y                              N
Street 2        20 A Y                              N
Street 3        20 A Y                              N
Town            15 A Y                              N
County          10 A Y                              N
Postcode         9 A N                              N
Telephone       12 A N                              N
Region           2 A Y                              N
Interests       25 T Y                              N
Notes           25 T Y                              N
NAME             0                                  
ADDRESS          0                                  
```

L Space 28883
Record format

Prompt

```
Prompt.......................................Value list...........
Title: Mr, Mrs, Ms, Dr, Sir, Lady            Mr,Mrs,Ms,Dr,Sir,Lady
First name: Bridget or B.
Surname:
Membership: S)ocial, T)ennis, H)onorary      S,T,H
Membership number:
Fees paid: N, Y                              N,Y
Paid up to:
Address line 1:
Address line 2:
Address line 3:
Town:
County:
Postcode:
Telephone number:
Region: SC, NE, NW, M, WW, S, SE, EA         SC,NE,NW,M,WW,S,SE,EA
Interests:
Further notes:
Dummy field
Dummy field
```

Figure 11.2. Format for name and address database.

The Interests section is marked as T, a text field, so the SELECT utility can search for individual words within the field. If it were alphanumeric, then searching could only be done for each person's first listed interests.

Finally, a coding system is used for a few of the fields. Y and N are obvious, but the Region field and type of membership are also coded. This does two things; first it saves memory because 'WW' is shorter than 'Wales and West', and second it standardises the data. All possible codes are listed in the Value list section. Note you must have commas between the values in this list; if there are extra spaces then the checking still works properly. The field width for the Region field is limited to two characters, because the region codes are only one or two characters long.

However, one potential problem arises here. When the Value list is:

```
SC,NE,NW,M,WW,S,SE,EA
```

it is still possible to type in 'SC,NE', because SC and NE are adjacent in the Value list. But 'SC,NW' and 'NE,SC' can't be entered, and nor can 'SC NE'. To prevent this happening, the Region field is limited to two characters and scrolling is disabled, so there is no room to type in such a code. For the same reason, the Membership field is limited to a single character.

In some circumstances, it is possible that this property might come in useful to denote people who belong on the border between two categories. But the Value list can get too complex to allow all the possibilities.

Entering the Data

This is often easiest using the Card mode; the display is certainly familiar to anyone who has filled in a few ordinary file cards. Arranging the fields on the card is just a matter of putting them together in sensible looking groups. Figure 11.3 shows one possible card layout. Depending on how often you'll want to browse through the database, it may be useful to increase the width of the Interests and Notes fields, so that more of the information can be seen on one card; there is plenty of room for this.

Card layout

<pre>
        NAME   ADDRESS

      Title _____   Street 1 _____
 First name _____   Street 2 _____
    Surname _____   Street 3 _____
                               Town _____
                             County _____
                           Postcode _____
 Membership _
     Number _____   Telephone _____
       Paid _
Date next due _____   Region _

         Interests _____

            Notes _____
</pre>

Figure 11.3. Card layout for membership list.

The only important guideline for designing the card is that the fields should be grouped logically, all the address fields together, one below the other, for example. Notice that there are two 'headings' on the card layout, 'NAME' and 'ADDRESS'. These are *dummy fields*, the last two fields in the record format, and they are assigned a width of zero. Having no width, they don't appear at all on the Spreadsheet display. But on the card display, the fieldnames do appear, and can be used to annotate the card. They can be picked up and placed just like any other field. No data can be put into these dummy fields as long as their width remains zero in the record format. The field cursor can't even be moved to them.

If the database is going to be big, just typing the data in is a big task, so make sure that the card layout is not going to be a problem. The best way to do this is to set everything up, then type in some records, perhaps 15 or so. An inconvenient card will soon show itself up, and the layout can be changed as a result. It's much harder to change the order of the fields in the record format, but it usually can be done. This is quite complex and will be described later.

When a few trial cards have been filled in, it's time to try out some of the intended uses of the database. It is better to try now than to find out later that an important facet of the information has been left out of the database. If all is well after the trial run, then the rest of the data can be

collated and entered into ViewStore. But don't be afraid to start again if something has been forgotten.

When All the Data's In

All the information has been entered, so what now? A database is a resource, not just a store, and getting data out in a convenient form is part of the justification for all the effort spent typing the data in.

Mailshots and Macros

At least once a year, a club maintaining a list of members has to send out reminders that the subscription or fee is due. This can be done by writing a standard letter using a wordprocessor such as VIEW, then sending the letter to all the people who haven't paid up. But VIEW has the capability to personalise each of those letters, using the *macro* facility. The letter may look something like that in figure 11.4. By surrounding this letter with DM and EM commands as shown, it becomes a *macro definition*. In this case, the macro is named 'AA', the name appears after the DM command. The letter isn't complete: it has gaps or *patterns*, marked by @0, @1 and so on.

```
DM AA
LJ @0 @1 @2
LJ @3
LJ @4
LJ @5
LJ @6
LJ @7, @8
LJ 1 August 1987

    Dear @0 @2,

    Can we remind you that your subscription to the North
    Humberside Real Tennis Federation is due once again.
    Owing to the ever-increasing cost of court
    maintenance, the Committee has reluctantly been forced
    to raise the annual subscription rates to £20 for
    social members, and £135 for playing members.
    However, we have now made reciprocal arrangements with
    the South Humberside Federation, and they have made
    available to us their court and changing facilities at
    Garthorpe.

    Yours sincerely
LJ Ron Shepherd
LJ Secretary
PE
EM
```

Figure 11.4. VIEW standard letter as macro definition.

157

The macro facility means that wherever the patterns @0 or @1 appear in the letter, they can be replaced in the printed version by something else, supplied in the form of a *macro parameter list* or *macro call*. This is a list like this

```
AA Dr,E. F.,Connolly,The Surgery,<5,Maxwell Close>,,Beverley,North Humberside,HU21 7FE
```

The various patterns in the definition are numbered @0, @1, @2 and so on, up to a possible @9. 'Dr' replaces @0 because it is at the beginning of the list; it appears wherever @0 appears in the macro definition in figure 11.4. 'E. F.' is @1, so it appears where @1 appears, and so on. The parameters are separated by commas.

Notice that the @5 parameter is blank and has just the two commas. This will create a blank line in the standard letter, with only an LJ command on it. Interestingly, VIEW doesn't print this blank line; unless there is something after the LJ, it just ignores it. This is good, because it means that addresses like Dr Connolly's, which have only two lines before the town name, don't print with a gap in them. It is also worth looking at the @4 parameter:

```
<5, Maxwell Close>
```

Although this has a comma in it, it's still all contained within one parameter held together by the angle brackets.

The completed personalised letter is shown in figure 11.5, with all the patterns replaced by their respective parameters.

```
Dr E. F. Connolly
The Surgery
5 Maxwell Close
Beverley
North Humberside, HU21 7FE
1 August 1987

Dear Dr Connolly,

Can we remind you that your subscription to the North...
```

Figure 11.5. VIEW standard letter with parameters replaced.

Macros are ideal for sending out lots of standardised letters like this, and there is a special ViewStore MACRO utility aimed at extracting macro parameter lists from databases. These lists can then be combined with a VIEW macro definition such as figure 11.4, and printed out. One

parameter list can be produced from each record in the database, so a personalised letter can be produced for every club member.

The MACRO Utility

To use the MACRO utility the database must be loaded as with other utilities. As usual, if using a BBC B micro, it's best to change to a less memory-hungry mode. And use the LIST command to get a list of the fieldnames.

Then run the utility as follows:

```
UTILITY MACRO
```

The utility prompts for four pieces of information. First, it asks whether to use a selection file. Clearly, if you want to send the standard letter to everyone in the database, then the answer is no. But usually, the mailshot is more selective, and the selection must be done beforehand. Press ESCAPE to stop the utility at this point if necessary.

The second prompt is for the list of fields from which to take the parameter information. In the case of the club members database, these might be the Title, First name, Surname, Street 1, Street 2 fields, and so on. Just press RETURN on its own to terminate the list. Remember that the fieldnames can be abbreviated with wildcards: Ti*, First*, Sur*, Str*1, Str*2 will do.

Each macro pattern must equate with one database field. Confusingly, the macro patterns in VIEW are called @0, @1 and so forth up to @9, but the MACRO utility prompts for Field 1, Field 2 and so on:

```
Field 1? Title
Field 2? First name
Field 3? Surname
Field 4? Street 1
Field 5? Street 2
Field 6? Street 3
Field 7? Town
Field 8? County
Field 9?
```

Field 1 becomes @0, Field 2 becomes @1, and Field 10 becomes @9. In fact, MACRO will continue to prompt for field names until only RETURN is pressed, but remember fields beyond Field 10 can't be used by VIEW.

159

The MACRO utility then asks for the name of the macro - this is the two–letter name that appears after DM in the VIEW macro definition. In figure 11.4, this was 'AA':

```
Two letter macro? AA
```

Finally, the utility needs the name of a file into which to put the macro calls. No prefix is applied to this name, so if it should be in a particular directory or on a particular drive, then name the file explicitly:

```
Output file name? :2.V.SUBSMAC
```

ADFS, Net: A much longer directory name can be used if necessary, for example, '&.VIEW.CLUB.SUBMACRO'. This creates the file SUBMACRO, two levels down the directory tree from the root. The root is the main directory selected with *DIR.

Don't worry about fields with commas; the utility sorts them all out automatically, putting angle brackets in wherever necessary. The output file can be loaded into VIEW, and it may look like figure 11.6. In creating the macro file, ViewStore reports how many records it looks through, four in figure 11.6.

```
AA Dr,E. F.,Connolly,The Surgery,<5, Maxwell Close>,      ,Beverley,Humberside,HU21 7FE
AA Mrs,Bridget,Cartwright,Top Flat,17a York Road,        ,Pocklington,N. Yorks,YO17 5FT
AA Mr,Ernest,Darby,21 Temperance Villas,          ,       ,Filey,N. Yorks,FI4 7LH LS|A
AA Ms,Carol,Folkley,c/o Selsdon Hall,Hull Univ.,Cottingham,nr. Hull,Humberside,
```

Figure 11.6. A VIEW MACRO file.

Notice that a field that is blank in the database causes a blank parameter consisting of eight spaces: this won't usually be a problem. But if it is, then the blanks can be deleted by loading the file into VIEW VERSION 3, and using the command:

```
CHANGE    ,^S, ,,
```

Sending Out a Subs Mailshot

There are several stages involved in sending out the actual mailshot, a process often referred to as *mailmerging*. The first process is to think about to whom the letters should go. Broadly, these are the people who have not paid their subscriptions yet. But a wise club secretary sends an encouraging letter to those whose subs will be due within a month, and a different letter to those whose subs are now overdue. Why? Because

160

the people with overdue subscriptions all got the encouraging letter the previous month!

When the recipients can be identified, go ahead and write the letter using VIEW. It might be easiest if the letter is written to one particular member, with all the personal data in place. Then at the end, take out all the personal information and replace it with the macro patterns: @0, @1 and so on. Put in the DM and EM commands, and give the macro a name, say AA. At this stage, note down the name, and what each of the macro patterns is: @0 is the Mr or Mrs, @1 is the first name, and so on.

The next stage is selection. If the macro is written so that the salutation is 'Dear Dr Connolly' or 'Dear Ms Cartwright', then it can go to anyone; but if it is designed around 'Dear Bridget', then a separate selection for all those people whose first name is unrecorded must be carried out, to eliminate the 'Dear E. F.' problem we have already come across in this chapter.

Switch to ViewStore and load the database, then run the SELECT utility. The selection criteria might look something like this:

```
Date next due < 1.8.87
AND Date next due >= 1.7.87
AND First name <> ?.*
AND Membership <> H
```

This puts all those whose subs are due during July, whose full first name is known, and whose club membership is not honorary, into the selection file. This file controls which of the database records is used to create a macro parameter list, and ultimately who gets a letter. It's convenient to sort the records into alphabetical order at the same time.

Next run the MACRO utility, using the selection file. List each of the required fields, by referring to the notes, and remember that Field 1 is really @0. ViewStore will create a macro parameter file.

Still in ViewStore, load a printer driver if necessary, and use the LABEL utility to print labels for all the letters to be sent out. Just use the same selection file; it doesn't have to be re-selected.

Finally, return to VIEW. Load in the macro definition file containing the standard letter. At this stage, it is worth checking a few things. Does the definition have a page eject command, PE, at the bottom to ensure

that each successive letter starts on a fresh sheet of paper? Are the margins set correctly? Are there any unneccessary blank lines?

When everything looks just right, read in the macro parameter file using the READ command. By using READ, the parameter lists created by ViewStore get added to the end of the definition text. It is best to preview a few of the letters, by using the SCREEN command, to check that the correct data is being substituted for the macro patterns @0, @1 and so on. Then load the correct printer driver, with the PRINTER command, and print out the letters.

As the selection file was in alphabetical order, the letters will be printed in the same order as the labels, so it should be straightforward to marry the two together.

Line Length Problems

There is a single problem with the process outlined above. Details for each letter can be taken from the database and inserted where @0 and @1 appear in the macro definition. But an entire macro parameter list must fit on one line, and VIEW lines can't be longer than 132 characters. What if the name, address and so on, add up to more than 132 letters? A message similar to:

```
3 fields split
```

indicates the MACRO utility has created three macros with parameter lists of more than 132 characters. The MACRO utility splits each over–long line into two.

The ViewStore manual suggests that a split list should be abbreviated in VIEW so as to fit on a single line. Fine, if only one or two macros overflow the limit, but what if 100 or 200 do? If you keep a database containing addresses of legal or financial partnerships, or a database of advertising agencies with their long intricate names, it will happen all the time! Chapter 16 presents an answer to this.

File Maintenance

So what happens when the subscription money rolls in? The database has to be updated to show people have paid, and when they are next due to pay.

The simplest way to do this is to load the database and go to the Card mode display, and then use the Surname index to sort out the records into order:

```
f6
Fields: Surname
Surname
```

Then any record can be found quickly using the LOCATE function (key f7):

```
f7
Value?
Cart*
```

will find the first person with a name starting 'Cart'. The new details can be entered onto the card, and the next person searched for.

At the end of the session, pressing ESCAPE to return to Command mode ensures all the data is saved and up-to-date.

Deleting Records

Inevitably with a club membership, new people join all the time, and members also leave. On the database, new records are added to the end of the data file, and the file lengthens. Old records are deleted from the middle of the database, but ViewStore can't 'close up' the gap left; the data file never gets shorter because the space taken up by a deleted record is not reused. So even if the number of records in a database stays about constant, the file will grow and grow. And ViewStore will consequently work more slowly and require more disc space than necessary for the files.

Periodically, it is an advantage to reclaim the disc space used up by these deleted records. There are two ways of doing this. The first relies on ViewStore's CONVERT utility. This always works by creating a new database, with information taken from the original. But this information can be modified in various ways.

First, the amount of spare space in each record can be increased or decreased. This spare space is the 'Space' figure in the top left corner of the data screen. If the information in a record is changed, then that record may run out of room. New space can be created with the CONVERT utility.

Second, a new sub-database can be created containing only some of the records from the main database. A selection file is used to select which records should be incorporated into a new database.

Third, the fields within the records can be stored in a different order. For example, the membership database could be re-ordered so that the first three fields were Surname, First name and Title instead of Title, First name and Surname. However, this also entails creating a new format file for the new database; the old format is not good enough because the fields are in the wrong order.

Fourth, the space used by deleted records can be reclaimed; they are not copied over into the new database.

Running the CONVERT utility, the prompts are:

```
Use select file (N,Y)?
```

Enter 'N' unless you want to create a sub-database, in which case only the records in the selection file are carried over into the new database. These records must be selected beforehand.

The next group of prompts begins with:

```
Field 1? *
Field 2?
```

To copy all the fields in the original order and merely reclaim the deleted record space or regulate the spare space in each record, just type the '*' wildcard as shown. This is the most usual response. Otherwise, enter the names of the fields to be carried over, in the order they should appear in the new database. End the list with RETURN on its own. If the fields are to be rearranged, it is a good idea to plan this out in advance. Before using CONVERT, print the names of the fields in the database. Make sure the printer is connected and type:

```
LIST CTRL-B
CTRL-C
```

Use the list to plan the new order of fields.

Next, CONVERT asks for the new record size. This is specified in exactly the way it is in the database header, explained in Chapter Nine. A figure of '+20' means that 20 extra bytes of space will be left, whatever the

length of the information. That leaves room for about 20 extra
characters of information, should the record need editing again. A
figure like '200' means that a minimum of 200 bytes is used for every
record, no matter how short it is. A 100 character record would have
about 100 bytes of free space; a record with 300 characters of
information would not be truncated, but no free space would be left.

Finally, the name of the database has to be entered: make sure it is
different from the old name! The appropriate prefix is used to name the
file, so:

    New file name? **NAMES**

could create a file called ':1.D.Names'. If the prefix shouldn't be used,
then enter a fuller filename, such as ':0.$.Names'. Creating a new
database in this way may take some time. When it is finished CONVERT
shows how many records there are in the new database.

Once a database is converted in this way, if the order of the fields has
changed, then a new format file will need to be created by editing the old
one. The new database, 'Names', can be loaded, using the old
database's format file, 'Members', with the command:

    LOAD NAMES F.MEMBERS

Furthermore, the old indexes will be out of date. New index files must
be created with the the INDEX utility described in Chapter Nine.

Purging Big Databases

A difficulty arises with converting large databases; there may not be
room on the disc for both the old and the new database files. The PURGE
program at the end of this chapter can reclaim the deleted record space
without the need for two copies of the database. Note that this is a BASIC
program, not a ViewStore utility.

To use PURGE, type in the BASIC listing, and save it. Run the program, and
it prompts for the ViewStore prefix that was used when the database
was set up. If there was no prefix, then just press RETURN. Next, type in
the name of the database file to be purged. If just a simple file-name is
entered, then the prefix is added to this in the same way as ViewStore.
If a specific disc or directory is named, then the prefix is not added; so, if

the prefix is ':1.', then 'Members' becomes ':1.D.Members', and ':0.Members' remains ':0.$.Members'.

The program works through the file slowly, compacting it and removing the deleted records. When finished, the number of records removed is displayed on screen.

The PURGE program incorporates a measure of security; it checks to see if the file named is a ViewStore database before trying to compact it. This can't be fully foolproof, but it checks the file in the way used by ViewStore itself. Also, ESCAPE is disabled while the database file is being altered. However, if BREAK is pressed while PURGE is working, then the database will be corrupted. For this reason, you should always keep back-ups of the database files.

Keeping a back-up is also vital because the listing contains some assembly language, which may be difficult to debug. However, there is a simple check routine.

The PURGE Program

For the technically minded, PURGE checks that the file is a ViewStore database file by reading the last part of the file backwards. The structure of ViewStore files means that the last useful byte must be a CHR$ 1, the end of data marker. After this there may be some padding bytes, CHR$ 0, but nothing else. So reading backwards, if anything except padding bytes is read before the end of data marker, then the file is not a ViewStore database.

Once checked, the assembly language section reads chunks of the file into a 256-byte input buffer using the OSGBPB call. Then providing it isn't part of a deleted record, each byte is transferred to an output buffer. When the input buffer is empty, the next 256 bytes are read in from the file. When the output buffer is full, then it is written back to the file. If a record has been deleted, then the output takes up less room than the input did. The input and output pointers progress through the file together, with the output pointer trailing further behind as deleted records are encountered.

If using an early BBC micro fitted with Basic 1, then alter line 2320 to read as follows:

```
2320 IF control$ = "U" THEN handle% = OPENIN name$
```

Basic 1 can be identified by typing:

```
*BASIC
REPORT
```

The date '1981' is displayed for Basic 1.

Program Listing 11.1

```
10 REM PURGE Viewstore deleted
20 REM record remover
30 REM for B/B+/M/C/2P/E + Viewstore
40 REM (C) Graham Bell 1987
50 :
60 REM Viewstore file constants
70 space% = &03
80 end_of_field% = &09
90 end_of_record% = &0D
100 end_of_data% = &01
110 deleted% = &02
120 pad% = &00
130 :
140 ON ERROR PROCerror (0)
150 PRINT "PURGE"
160 :
170 prefix$ = FNname ("Prefix for data files")
180 name$ = FNname ("Name of main database")
190 IF INSTR(name$, ".") = 0 THEN name$ = prefix$ + "D." +
    name$
200 IF NOT FNconfirm ("Have you made a backup") THEN GOTO 330
210 :
220 big_file% = FNopen (name$, "U")
230 big_eod% = FNeod_pointer (big_file%)
240 :
250 PRINT ' "purging";
260 *FX 229,1
270 records% = FNcompact_file (big_file%)
280 IF POS <> 0 THEN PRINT
290 PRINT ; records%; " deleted record(s) purged"
300 *FX 229
310 :
320 CLOSE# big_file%
330 END
340 :
350 :
360 :
370 DEF FNcompact_file (file%)
380 LOCAL A%, code%
390 code% = FNassemble
400 A% = file%
410 = ((USR code%) AND &FFFF00) DIV 256
```

```
420 :
430 DEF FNassemble
440 LOCAL I%, P%, sum%
450 osbyte = &FFF4
460 osasci = &FFE3
470 osgbpb = &FFD1
480 :
490 DIM addr &2FF
500 FOR I% = 0 TO &2FF
510 I%?addr = 0
520 NEXT
530 :
540 FOR pass = 0 TO 2 STEP 2
550 P% = addr
560 [        OPT pass
570          STA ipctrl
580          STA opctrl
590 .main    JSR fill
600 .main_1  LDX ipptr
610          LDA ipbuff,X
620          INC ipptr
630          CMP #deleted%
640          BEQ main_3
650          CMP #end_of_record%
660          BNE main_2
670          LDX lastb
680          CPX #deleted%
690          BNE main_2
700          PHA
710          LDA #46
720          JSR osasci
730          PLA
740          INC count
750          BNE main_3
760          INC count + 1
770          JMP main_3
780 .main_2  LDX opptr
790          STA opbuff,X
800          INC opptr
810          BNE main_3
820          PHA
830          JSR empty
840          PLA
850 .main_3  STA lastb
860          LDA ipptr
870          BNE main_1
880          BIT eoflag
890          BPL main
900          LDA opptr
910          BEQ main_4
920          JSR empty
930 .main_4  LDX count
940          LDY count + 1
```

```
 950         RTS
 960  :
 970 .fill   LDA #0
 980         LDX #1
 990         STA ipctrl + 8
1000         STA ipctrl + 7
1010         STX ipctrl + 6
1020         STA ipctrl + 5
1030         LDA #130
1040         JSR osbyte
1050         STY ipctrl + 4
1060         STX ipctrl + 3
1070         LDA #ipbuff DIV 256
1080         STA ipctrl + 2
1090         LDA #ipbuff MOD 256
1100         STA ipctrl + 1
1110         LDA #3
1120         LDX #ipctrl MOD 256
1130         LDY #ipctrl DIV 256
1140         JSR osgbpb
1150         ROR eoflag
1160         BPL fill_2
1170         LDA #0
1180         SEC
1190         SBC ipctrl + 5
1200         TAX
1210         LDA #deleted%
1220 .fill_1 STA ipbuff,X
1230         INX
1240         BNE fill_1
1250 .fill_2 RTS
1260  :
1270 .empty  LDA #0
1280         LDX #1
1290         STA opctrl + 8
1300         STA opctrl + 7
1310         STA opctrl + 6
1320         LDY opptr
1330         BNE empt_1
1340         STX opctrl + 6
1350 .empt_1 STY opctrl + 5
1360         LDA #130
1370         JSR osbyte
1380         STY opctrl + 4
1390         STX opctrl + 3
1400         LDA #opbuff DIV 256
1410         STA opctrl + 2
1420         LDA #opbuff MOD 256
1430         STA opctrl + 1
1440         LDA #1
1450         LDX #opctrl MOD 256
1460         LDY #opctrl DIV 256
1470         JSR osgbpb
```

169

```
1480          RTS
1490  :
1500  .ipptr  OPT FNequb (0)
1510  .opptr  OPT FNequb (0)
1520  .eoflag OPT FNequb (0)
1530  .lastb  OPT FNequb (0)
1540  .count  OPT FNequw (0)
1550  :
1560  .ipctrl OPT FNequb (0)
1570          OPT FNequd (0)
1580          OPT FNequd (0)
1590          OPT FNequd (0)
1600  .opctrl OPT FNequb (0)
1610          OPT FNequd (0)
1620          OPT FNequd (0)
1630          OPT FNequd (0)
1640  :
1650  .ipbuff OPT FNequs (STRING$(128, CHR$ 0))
1660          OPT FNequs (STRING$(128, CHR$ 0))
1670  .opbuff OPT FNequs (STRING$(128, CHR$ 0))
1680          OPT FNequs (STRING$(127, CHR$ 0))
1690  .end    BRK
1700  ]
1710  NEXT
1720  :
1730  IF end - addr <> &2F8 THEN PROCerror (3)
1740  = addr
1750  :
1760  DEF FNequb (byte%)
1770  ?P% = byte%
1780  P% = P% + 1
1790  = pass
1800  :
1810  DEF FNequw (word%)
1820  ?P% = word% MOD 256
1830  P%?1 = word% DIV 256
1840  P% = P% + 2
1850  = pass
1860  :
1870  DEF FNequd (double%)
1880  !P% = double%
1890  P% = P% + 4
1900  = pass
1910  :
1920  DEF FNequs (string$)
1930  $P% = string$
1940  P% = P% + LEN string$
1950  = pass
1960  :
1970  DEF FNeod_pointer (file%)
1980  LOCAL ptr%, byte%
1990  ptr% = EXT# file%
2000  REPEAT
```

```
2010 ptr% = ptr% - 1
2020 PTR# file% = ptr%
2030 byte% = BGET# file%
2040 UNTIL (byte% <> pad%) OR (ptr% = 0)
2050 REM only eod marker may preceed padding in valid file
2060 IF (byte% <> end_of_data%) OR (ptr% = 0) THEN PROCerror(2)
2070 = ptr%
2080 :
2090 DEF FNname (prompt$)
2100 LOCAL name$
2110 PRINT ' prompt$;
2120 INPUT "? " name$
2130 REM delete trailing spaces
2140 REPEAT
2150 IF RIGHT$(name$, 1) = " " THEN name$ = LEFT$(name$, LEN
     name$ - 1)
2160 UNTIL RIGHT$(name$, 1) <> " "
2170 = name$
2180 :
2190 DEF FNconfirm (prompt$)
2200 LOCAL key$
2210 PRINT ' prompt$;
2220 INPUT " (N,Y)? " key$
2230 key$ = LEFT$(key$, 1)
2240 = INSTR("Yy", key$) <> 0
2250 :
2260 DEF FNopen (name$, control$)
2270 REM requires Basic 2
2280 LOCAL handle%
2290 control$ = LEFT$(control$, 1)
2300 PRINT "Opening "; name$
2310 IF control$ = "R" THEN handle% = OPENIN name$
2320 IF control$ = "U" THEN handle% = OPENUP name$
2330 IF control$ = "W" THEN handle% = OPENOUT name$
2340 IF handle% = 0 THEN PROCerror (1)
2350 = handle%
2360 :
2370 DEF PROCerror (number%)
2380 *FX 229,1
2390 CLOSE# 0
2400 IF number% = 0 THEN REPORT : PRINT
2410 IF number% = 1 THEN PRINT "Can't find file"
2420 IF number% = 2 THEN PRINT "File not a database"
2430 IF number% = 3 THEN PRINT "Assembler error - please check
     listing"
2440 *FX 229,0
2450 END
```

Listing 11.1. PURGE deleted record remover.

12 : Diverse Reports
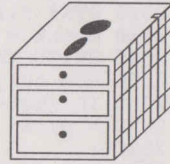
The biggest single advantage of a large electronic database is the ability to create a summary of its contents quickly. This can't be done with a traditional card-based file because you would have to search through all the cards manually. Of course, the computer can do this searching, and it can sort the cards, or records, into order too, using the SELECT utility. The next requirement is to produce a summary of the information in the selected and sorted records. ViewStore does this with the REPORT utility.

Producing simple lists from a selection file using the REPORT utility was described in Chapter 10. In a way, this is a summary of the records in the selection file, but it is always the first few fields in each record that get printed. But REPORT offers much more sophisticated control over the summary, by using a *report definition file*.

The report definition file specifies which of the selected record fields should be displayed, and also how each field is shown. Mathematical operations can be carried out on numeric fields, and totals and subtotals of particular fields can be be printed. The report file can also control the page layout of a printed report.

Setting Up a Report Definition File

In fact, a report definition is another mini-database: it consists of the report file itself, and a special report format file F.REPORT. All report files use the same F.REPORT file. This special file is supplied on the ViewStore utilities disc, and should be copied to any working database disc or directory along with the utility files. It should be on the disc along with any other format files, so the same prefix can be used. For example, with two double-sided disc drives, the format files are usually put on drive two, and the prefix set to ':2.'.

Just as with a complete database, the SETUP utility is used to create a new report file. The first prompt from this utility is:

```
Create database or report (D,R)?
```

You should answer 'R' to make a report file. The utility then asks for the name of the report file. It's possible to use the same name as for the other database files; in this case D.BIBLIOG and F.BIBLIOG would be joined by R.BIBLIOG. But it is also possible to have a range of different report files, to create reports on a number of aspects of the same data. Clearly, each report file needs a separate, recognisable name.

DFS: One point to be careful of when using the DFS is that the same 'D' prefix is used for creating database files and report files. So, if a report is created, it usually gets put on the disc after the end of the database file. This may leave the database file little or no room to grow as new records are added to it, and eventually the dreaded 'Can't extend' error message will occur. To avoid this serious problem, it is better if the new report file is created on a different drive, perhaps where the utility files are kept. For example, if the database file is in directory 'D' on drive zero and the utilities are on drive one, then the report file called MONTHND can be created by using the full name:

```
Name of report? :1.R.MONTHND
```

If the full name is used like this, then the prefix is ignored. Henceforth, because the standard 'D' prefix won't apply to the report file, the full filename will have to be used. This whole process is unnecessary for ADFS or on a network, because the 'Can't extend' error doesn't occur.

Before using the report file to produce a report, it has to be filled with data about how the report should look. To do this, the report file is treated like an ordinary database. A report file called MONTHND can be loaded into ViewStore by entering the following:

```
LOAD R.MONTHND REPORT
```

The report file is always in directory R, and R must be typed in; if it isn't, then ViewStore expects to load a database file from the D directory,

173

and of course, D.MONTHND doesn't exist. Typing in REPORT means that ViewStore uses the filename F.REPORT with the appropriate prefix, to load the special report format file.

DFS: The full report filename' :1.R.MONTHND' should be used if the file was created as above.

The Report Definition

Pressing ESCAPE after loading the new report file presents a screen like that in figure 12.1; this is the *report format definition*, but its structure resembles that of a database. There may be any number of rows in this database. Each has just four columns known at 'T', 'Half1', 'Half2' and 'Field list'.

```
   L Space 300    Indexed by entry
                             Report format definition
   Type(C=comment,H=header,M=margin,P=pagelength,R=record,S=subtotal,T=total)

   T.Half1.................................................................
```
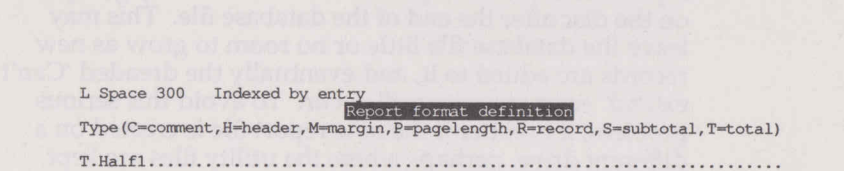
Figure 12.1. Initial blank report definition.

Layout Types

The T field marks the *line type*. There are seven different types of report line, and the prompt for the line type codes is shown in figure 12.1. The simplest line types are the two that control aspects of the page layout: P and M.

A P-type line changes the page length of the report. If continuous A4-sized paper is used to print the report, then the page length should be set to 70. There is room for 70 lines per sheet with a normal printer (70 lines of a sixth of an inch is $11\frac{2}{3}$ inches). The page length is put in the 'Half1' field:

```
   T.Half1.........................................
   P 70
```
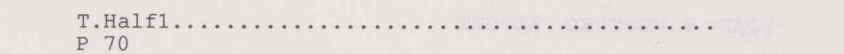
It won't necessarily look like this on the screen. When the cursor is

moved to the 'Half1' field, the screen scrolls, so the T field can't really be seen. Putting in more than one P line is pointless, as only the first is obeyed. However, this line can be anywhere in the definition. Without a P line, a page length of 66 is assumed, and this is a suitable length for most normal listing paper.

An M line controls the number of blank lines left as a margin at the foot of each page of a printed report. The default margin is four blank lines, which usually isn't enough. It can be increased in the same way as the page length:

```
T.Half1......................................
M 10
```

Record and Header Types

The bulk of a printed report should be made up of *record lines*. These lines' T field contains 'R'. As the report is printed, the record line is printed for each record in the database. Of course, this can be limited to one for each record in the selection file.

Field Half1 of a record line contains the text that should be printed for each of the records in the database. Within the text, there may be gaps marked by the special @ character. These gaps, or *patterns* or *formats*, can be replaced by information drawn from each of the records in the database while the report is printed.

Header lines, type 'H', contain the text that should appear at the top of each page of the printed report. For example, if each page should be headed by a title giving the name of the report, then that name should be typed onto a header line. In fact, just like record lines, header lines can contain patterns too, and information can be drawn from the first record on each page to fill them.

But let's start simply, try setting up a report definition like that shown in figure 12.2. First use the SETUP utility to create a report file called, say, FIRST. Load it with:

```
LOAD  R.FIRST  REPORT
```

Then type in the text shown in the figure. Notice that there are two header lines. The first contains text and a pattern; the second is blank.

This means that at the head of each page a line is printed first with some text and then a blank line.

```
T.Half1.............................................
P 12
H header line for page @@@@
H
R record line for record @@@@
```

Figure 12.2. A simple report demonstration.

What should fill in the two patterns in the H and R lines? Put the cursor on the first of the header lines, and press RIGHT. The 'Half2' field is scrolled on. Press RIGHT again, and the 'Field list' field appears. This is a list of the information that should be printed instead of the patterns. In this simple example, type '|P' into the field list. |P is a *register*, a number that the REPORT utility remembers. In fact there are 50 of them, but two have special tasks. |P is the page register, it counts the pages in the report as it is printed. VIEW has a similar page register. |R is the record register counts through the records one by one. Put |R in the field list for the report line. (Note that '|' is SHIFT-\). The completed field lists should look like figure 12.3.

Once the report definition is complete, pressing ESCAPE returns to Command mode and saves the details of the report definition in the report file.

```
Field list....................................
|P
|R
```

Figure 12.3. Report demonstration field list.

Printing a Report

To print a report, the database must be loaded. Any database is suitable for the simple demonstration using the report definition in figure 12.2, but in particular the ViewStore 'CARS' example database or the CREDIT database on the disc accompanying this book may be used.

First, load the database, and use the SELECT utility to make any selection necessary. For example, selecting all the records in the database in which a particular field has been filled in:

```
Select criteria? Amount <> ""
```

The double quotes denote a blank field.

Once the selection is made, change to a screen mode which leaves plenty of memory free, because reporting requires a large amount of space. Then run the REPORT utility. Answer the usual selection file prompt and then indicate whether the report should be printed out or merely shown on screen. If a printer is connected, then answer 'P'. Type 'S' to merely see the report on screen. The third prompt is shown below:

```
Use report format file (N,Y)?
```

Answer 'Y', then enter the name of the report file, FIRST. The rest of the prompts can be ignored by just pressing RETURN to give their default answers. The whole process is shown in figure 12.4, and the print-out of the second page from the report itself is in figure 12.5. This will look the same whatever database is used to generate the report.

```
=>LOAD  BIBLIOG
=>UTILITY  REPORT
REPORT
Use select file (N,Y)? N
Screen or Printer (S,P)? P
Use report format file (N,Y)? Y
Report filename? FIRST
Send totals to linking file (N,Y)?
Subtotal field?
Single sheets (N,Y)?
=>
```

Figure 12.4. Printing out the simple report.

```
header line for page       2

record line for record     7
record line for record     8
record line for record     9
record line for record    10
record line for record    11
record line for record    12
```

Figure 12.5. A single page of report print-out.

With the printout to hand, look back at the report file that created it.
First, on the page are the two header lines, one with text, one without.
The @@@@ pattern from the first header line has been replaced by the
page number taken from register I P. Following the header are a
number of record lines, one for each of the records in the database.
These are numbered because the value of register I R, the record
counter, is included on each line. Finally, at the bottom of the page are
four blank lines, the default margin.

Another detail can be learned from this simple report. Notice that there
are big gaps before the numbers. This is because the numbers are
ranged to the right within the pattern, so:

```
header line for page @@@@
```

becomes the following:

```
header line for page       2
```

A More Complex Report

The new report definition shown in figure 12.6 illustrates a number of
new points. It is designed to be used with the CREDIT database on the
Dabhand Guide disc, but the principles can be used with any database.
However, at this stage it is useful to have a printed list of the
fieldnames used in the database for which the report is intended.

Load the database, connect the printer, and type in the following:

CTRL–B **LIST** RETURN CTRL–C

```
L Space 296    Indexed by entry
                          Report format definition
Type(C=comment,H=header,M=margin,P=pagelength,R=record,S=subtotal,T=total)

T.Half1.................................................
H
H Page @@
H
R @@@@@@@@@@@@@  invoice @@@@@@@@@ of @@@@@@@@  £@@@@@.@@
R                    for @@@@@@@@@@@@@@@@@@@
M 10
P 66
```

Figure 12.6. Complex report definition example.

In figure 12.6, the most obviously new feature is that there are several
patterns on the record line, and that they are much bigger. These
patterns are designed to be replaced by information from the database
rather than the numbers from the registers. The field list is just what it
says: a list of fields from which to take information. So the field list for
the record line could be:

```
Company, Invoice, Date, Amount
```

The order of the fieldnames is important. The order the fields appear in
the list is the order they get used to replace the patterns. This means
that the record line:

```
@@@@@@@@@@@@@  invoice @@@@@@@@@ of @@@@@@@@  £@@@@@.@@
```

might eventually be printed as:

```
Systematic Pip  invoice    6781234 of  25.5.86  £    24.67
```

The Company information goes in the first pattern, the invoice number
in the second, and so on. It's important to make sure that the number of
items in the field list matches the number of patterns there are in the
record line's Half1 field, and that the different types of field match up
with their intended patterns.In printing out a report, if the error 'Not
enough fields' occurs, it really means there are too few patterns on the
stated line in the report definition. Too many patterns doesn't cause an
error; the excess patterns are merely ignored.

179

The various parts of the field list are separated by commas. The fieldnames in the field list must be typed in carefully. As usual with field names, the question mark (?) and star (∗) wildcards can be used, but unlike other uses of wildcards, any abbreviated field name must be enclosed in delimiters, like this:

```
Company, "Inv*", Date, "Amo*"
```

Delimiters are also needed if the fieldname contains a comma or other odd character, or even if the fiel name contains a space. Delimiters should preferably be double quotes (") or single quotes ('), other punctuation characters can be used if necessary.

As the record line is a mixture of text and patterns, some of the print-out is text printed literally, other parts are taken from the database. The length of the @@@@ pattern controls how much of the database field gets printed. In this case, there is room in the pattern for 'Systematic Pip'; the full company name is actually longer but is truncated because the pattern is only 14 characters long.

The final pattern contains a decimal point. This is only useful when the corresponding field is a numeric type, but it controls the number of decimal places used when the number is printed in the report.

In fact, there are two record lines in figure 12.6. The second has another pattern on it, for which the field list is 'Stores Item(s)'. This means that each record in the database will produce two lines of report, like this:

```
record line a for record 3
record line b for record 3
record line a for record 4
record line b for record 4
record line a for record 5
```

Again, it doesn't matter where within the report definition the two lines are, but the order they are printed in is the order they appear. There can be any number of record lines. It is sensible to keep them all together in the definition, or one may get overlooked.

Registers and Arithmetic

The REPORT utility is the only part of ViewStore that can carry out any sort of arithmetic. For example, if the database figures include VAT but

the invoice amount in the record line above should be quoted excluding VAT, then the field list could be amended to:

```
Company, Invoice, Date, Amount / 1.15
```

Then the pattern in the record line gets replaced by the newly calculated VAT-free invoice amount. The calculation is done separately for each record in the report.

Addition, subtraction and multiplication may be used as well as division, and brackets can be used to ensure that complicated expressions are calculated in the right order. Obviously, if a fieldname is used, then the corresponding field must be numeric! If it isn't, then a 'Type mismatch' error message will appear on screen. Sadly, even date fields can't be used, so time intervals cannot be calculated.

Calculation results can also be assigned to registers. Two registers have already been used, |R and |P, but these are the odd ones out. It is unwise to do anything with them except print them out, as in the simple report definition in figure 12.2. All the other registers, |A to |Z, are accumulators. Look at the following series of assignments. To start with look at:

```
A:10              makes  |A = 10
A:20              makes  |A = 30
A:5               makes  |A = 35
A:Amount / 1.15   makes  |A = 35 + (Amount / 1.15)
```

Each new figure assigned to |A doesn't replace the current value; it is added to it instead. 'A:1' doesn't make |A=1, it increments |A by one. This is quite unlike variables in BASIC or other programming languages, where the equivalent is:

```
A = A + 1
```

One perplexing source of errors is that you can't use spaces around the colon in a register assignment.

A Report With Subtotals and Totals

So what are registers used for? Most often, they are used to accumulate subtotals and totals of numeric fields. The report definition in figure 12.7 could use the following field lists for the two record lines:

```
Company, Invoice, Date, Amount, A:Amount
"Stores item(s)"
```

This uses the register |A to accumulate the total of all the invoices in the database. As each record line is printed, its invoice amount is added to |A.

```
L Space 296    Indexed by entry
                        Report format definition
Type (C=comment,H=header,M=margin,P=pagelength,R=record,S=subtotal,T=total)

T.Half1.....................................................
H
H Page @@
H
R @@@@@@@@@@@@@@   invoice @@@@@@@@@@ of @@@@@@@@ £@@@@@.@@
R                   for @@@@@@@@@@@@@@@@@@@@@@@@
S                                                 --------
S Subtotal owed                                   £@@@@@.@@
S                                                 --------
S
T -----------------------------------------------
T GRAND TOTAL OWED                                £@@@@@.@@
T -----------------------------------------------
M 10
P 66
```

Figure 12.7. Report definition with totals and subtotals.

To print out the invoiced total at the end, a *total line* is used. This has a T field containing 'T'. In fact there are three total lines, two being used just to highlight the real total. The total lines get printed right at the end of the report, after the record lines for the whole database have been finished.

There is a single pattern in the main total line, and just like a record line, it requires a field list. In this case, the field list would be as follows:

|TA

The pattern gets replaced by the value of |A when it is printed. And |A contains the total of all the invoice amounts. So:

```
T --------------------------------------------------------
T GRAND TOTAL OWED                            £@@@@@.@@
T --------------------------------------------------------
```

becomes the following:

```
T --------------------------------------------------------
T GRAND TOTAL OWED                            £ 1604.07
T --------------------------------------------------------
```

Figure 12.7 also used four *subtotal lines*, type 'S'. These can be printed out part way through the report. The way this works is as follows. The REPORT utility prompts for one or more subtotal fields. If a field name is entered, then ViewStore keeps an eye on the values in this field as the report is printed. Every time it changes, a set of subtotal lines is printed. The overall scheme may look like this, if the Company field were used as the subtotal 'trigger':

```
Ball & Co.
---Subtotal----
Systematic Pip
Systematic Pip
Systematic Pip
---Subtotal----
Therm-Ace Heat
Therm-Ace Heat
---Subtotal----
-----Total-----
```

Each accumulator register is really two separate registers: |TA provides the total stored in the |A register, and |SA provides the |A subtotal. The difference is that the subtotals are all reset to zero every time a set of subtotals are printed. So, just as total lines should contain patterns and field lists using |TA, |TB, or any other register, subtotal lines should use |SA, |SB and so on. |SA contains the total since the last set of subtotals.

Subtotals are only really useful if they are used with a selection file sorted into order. If the selection file was sorted using the Company field, then the selection trigger should be the Company field too. If the companies were not in order, then useless subtotals would be printed after almost every record.

The whole process of producing a report using the CREDIT database is shown in figure 12.8. The aim is to produce a summary of all the invoices received during one particular month. A selection file is created that contains all the records where the invoice date is during May 1986. This is then sorted into alphabetical order of company. Then the REPORT utility is used, with a report file like that in figure 12.7. The Company field is used as a subtotal trigger. Part of the printed report is also shown, in figure 12.9.

```
=>LOAD CREDIT
=>UTILITY SELECT
SELECT List or create select file (L,F)? F
Select criteria? DATE >= 1.5.86 AND DATE < 1.6.86
Select criteria?
Sort field? COMPANY
Sort field? DATE
Sort field?
Key width:9
Ascend or Descend (A,D)? A
Selecting.......................
13 records selected out of 24
SORT
Sorting....
=>UTILITY REPORT
REPORT
Use select file (N,Y)? Y
Screen or Printer (S,P)? P
Use report format file (N,Y)? Y
Report filename? :1.R.FIRST
Send totals to linking file (N,Y)? N
Subtotal field? COMPANY
Subtotal field?
Single sheets (N,Y)? N
=>
```

Figure 12.8. Generating a complex report.

```
Page   3

Systematic Pip   invoice DH981      of 21.5.86     £     35.89
for Polypropylene Hose
                                                   ---------
Subtotal owed                                      £    395.10
                                                   ---------

Therm-Ace Heat   invoice 00000004    of 8.5.86     £     76.29
                       for Copper Fittings
Therm-Ace Heat   invoice 00000005    of 13.5.86    £     12.35
                       for Moulded Elbows
Therm-Ace Heat   invoice 00000006    of 23.5.86    £      5.43
                       for Moulded Y Pieces
Therm-Ace Heat   invoice 00000007    of 27.5.86    £     17.11
                       for Moulded Attachments
Therm-Ace Heat   invoice 00000008    of 31.5.86    £      4.98
                       for Moulded Sundries
                                                   ---------
Subtotal owed                                      £    116.16
                                                   ---------
---------------------------------------------------
TOTAL OWED                                          £  1410.86
---------------------------------------------------
```

Figure 12.9. Part of printed report.

There is a major weakness with the subtotalling procedure. If the subtotal line contains any patterns and the field list refers to fields from the database, then the information comes from the first record after the subtotal, not the one before it. So a subtotal line like this:

```
S Subtotal owed to @@@@@@@@@@@@@@@@@@@@@@@         £@@@@@.@@
```

would produce a report like this:

```
Systematic Pip   invoice DH927      of 12.5.86     £    123.45
Systematic Pip   invoice DH981      of 21.5.86     £     35.89
                                                   ---------
Subtotal owed to Therm-Ace Heating                 £    395.10
                                                   ---------
Therm-Ace Heat   invoice 00000004    of 8.5.86     £     76.29
Therm-Ace Heat   invoice 00000005    of 13.5.86    £     12.35
```

Clearly the information on the subtotal line is wrong. To avoid this problem, field lists on subtotal and total lines should not refer to database fields, only to registers.

Number Formatting

The role of the @@@@ pattern is fairly clear with text fields; it controls the maximum length of the information in the report. If the database contains a field which is longer than the pattern, then the text is truncated. The pattern may only contain @ characters.

With numeric fields, the pattern can be more complicated. It may contain a decimal point, and it shows how many decimal places should be shown in the report. For example:

```
@@@.@@    shows 0 as            0.00
@@@.@@    shows 3.1 as          3.10
@@@.@@    shows 1,109,000 as    1.11E6
@@@       shows 6.1 as          6
@@.@      shows 1,109,000 as    %
```

The error in the last example is caused because the pattern calls for one decimal place. There is no room for 1109000.0, so the equivalent exponential form is used. However, there is no room for 1.1E6 either, so '%' signals an error. When reducing the precision of a number for display, ViewStore rounds properly, so 4.49 becomes four, whereas 4.50 becomes five and 1,109,000 becomes 1.11E6.

The last option in a numeric pattern is to use 'b' instead of the @ character:

```
@bb.bb    shows 3.1 as          3.10
@bb.bb    shows 1,109,000 as    1.11E6
@bb.bb    shows 0 as
@@@.@b    shows 0 as
```

The formatting works exactly the same, except that if the number is zero, then only blanks are printed. Only a single b is needed, and it can be anywhere in the pattern except the first character. 'B' can be used too. But it is conventional to use an initial @ and make the rest of the pattern from b characters.

Using Half2

Reports created in the usual way are limited to a width of 79 characters. This is because the Half1 field used to type in the report text and patterns has a width fixed at 79 characters, and it can't scroll. For wide

printers, or normal printers used with condensed mode print, another field, Half2, can be used to define what should be printed on the right-hand half of the paper.

Extra text or patterns typed into Half2 act in all ways as a simple extension of Half1. This can pose a couple of unexpected problems. Figure 12.10 shows two record lines, together with an example of how they would appear in a printed report.

```
Half1......................Half2.............
Invoice number @@@@@@      Date @@@@@@@@
Amount £@bbb.bb            VAT £@bbb.bb


Invoice number   1234Date  24.7.87
Amount £   11.50VAT £    1.73
```

Figure 12.10. Using Half2.

Notice that there is no automatic gap between Half1 and Half2 when the report is printed. 'Date' is printed out immediately after the number, filling the invoice pattern from Half1. And although VAT is aligned under 'Date' in Half2, they are not aligned when printed out, because their respective Half1 texts have differing lengths. To make this work, Half1 must be padded out with extra spaces, as shown in figure 12.11.

```
Half1......................Half2.............
Invoice number @@@@@@##     Date @@@@@@@@
Amount £@bbb.bb########     VAT £@bbb.bb


Invoice number   1234  Date  24.7.87
Amount £   11.50       VAT £    1.73
```

Figure 12.11. Using Half2 properly. Extra space = '#'.

The second problem can be caused when Half1 ends with a pattern and Half2 also begins with a pattern. Because there is no space between them, the REPORT utility treats them as a single, long pattern. Consequently, the last field in the field list has no pattern, and it never gets printed. The solution to this is to ensure that Half1 always ends with at least one space character.

Comment Lines

One of the major advantages of the ViewStore reporting system is that when it's decided what sort of a summary is needed, and the report definition file created, the same sort of summary can be done repeatedly just be re-running the REPORT utility. So every month, a series of month-end reports can be generated with the minimum of fuss. But then the report for April will look identical to the report for May.

Comment lines are a way of incorporating something unique into each report. A comment presents a prompt every time the REPORT utility is used, and the answer to the prompt can be printed in the report. For example, the report date could be printed at the head of each page.

A comment line is type 'C'. The prompt text can be put in the Half1 field:

```
T.Half1.....................
C Today's date?
```

The 'Today's date?' prompt becomes the last prompt that the utility asks before generating the report. Whatever is typed in response to the prompt becomes comment 1, or '^C1'. A second comment line with a different prompt could supply ^C2, a third ^C3 and so on.

The response can be printed in the report by including ^C1 in a field list; of course it needs a corresponding pattern too. A header line like this:

```
T.Half1........................Half2......Field list.....
H Summary dated @@@@@@@@@@@@@@@        ^C1
```

could be used to print the date of the report at the top of every page.

The responses to comments are always treated as plain text. Dates don't need to follow the ViewStore convention, so for example, '8th August 1987' can be used instead of '8.8.87'. Note that numbers typed in response to a comment prompt can't be used in any arithmetic.

Reporting on VIEW

The page layout features of the REPORT utility are fairly rudimentary. If greater control over the layout is required, if the report needs to be

annotated, or if it has to be incorporated into a larger overall document, then the report must be saved to a file, not printed.

The report can be sent to a text file using the VSXFER spooler described in Chapter Six. VSXFER is a printer driver, and it can be loaded into ViewStore with the PRINTER command before using the REPORT utility. When the report is created, a filename must be entered in answer to the 'Spool filename ?' prompt.

Eventually, this new file can be read into VIEW and treated as normal. Notes can be added, and all the page layout facilities of VIEW used to print an attractive report.

A tip worth remembering is that a spooled file really shouldn't be divided up into pages. If it is, then the extraneous blank lines and header lines might need to be removed when the file is read into VIEW. The REPORT facility normally does divide a report up into pages, with the header lines at the top of the page and the margin at the foot. Paging can be switched off by setting a page length of zero in the report definition, as below, so no extra lines are created:

```
T.Half1......................
P 0
```

Actually, this sets a page length of 256 lines, the maximum length that can be used. So for exceptionally long reports, a few extra lines may be created as a second or third extra-long page is required. However, this minimises the amount of editing that needs to be done once the report is transferred to VIEW.

ADVANCED

SECTION

13 : ViewSheet Hints and Tips
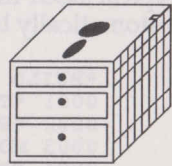
Using a !BOOT File With ViewSheet

When starting up ViewSheet, the same sequence of keypresses is often used to set the printer up, to select a favourite screen mode and colour, and so on. This whole sequence can be speeded up and made more convenient using an exec file.

An exec file is a file which is read in from disc or network, and simulates keypresses or commands entered at the keyboard. Any command in the file is executed just as if it had been typed in by hand. The most familiar example of an exec file is the !BOOT file that is often used to start up an application or game by pressing SHIFT–BREAK. This procedure is called auto-booting and the file itself is sometimes described as an auto-exec or auto-boot file.

Creating an Auto-exec File

The usual way to create an exec file is to type in the *BUILD command. But an exec file is really just a text file, so almost any wordprocessor can be used. Try to ensure there is no formatting of the text file, or any extra matter in it. The program to be careful with is InterWord, because it saves all the setup preference data at the beginning of each file, and so a normal saved file can't be used. The spooling option offered by InterWord and most other wordprocessors is ideal.

An ordinarily saved VIEW file is fine, providing there are no rulers, or edit commands. It is best to switch formatting off as well (with SETUP I on VIEW version 3 or later, otherwise press CTRL–f2), before starting to type in the file; this prevents any long lines being split up or justified.

The !BOOT file is a special type of file that can be executed automatically when SHIFT–BREAK is pressed. This could include all the commands regularly used when starting up ViewSheet. Here is how to create a

typical !BOOT file using *BUILD. The line numbers are provided automatically by the computer, and don't appear in the final file:

```
*BUILD  !BOOT
0001 *TV 0 1
0002 *SHEET
0003 MODE 3
0004 PRINTER JUKI4
0005 LOAD <filename>
0006 LW <windowfile>
0007 ESCAPE
```

This same text could be typed into VIEW or another wordprocessor just as easily, and saved in a file called !BOOT.

When executed, this !BOOT file switches off video interlace so that the display doesn't jitter up and down on a monitor, then selects ViewSheet and changes screen mode. Then it loads the printer driver, the spreadsheet model and a set of window definitions. Pressing ESCAPE completes entry of the *BUILD file.

The file can be executed at any time by typing:

```
*EXEC  !BOOT
```

but to make the file auto-boot, the following has to be entered too:

```
*OPT 4 3
```

This stores in the disc catalogue the information that the !BOOT file must be executed automatically, whenever SHIFT–BREAK is pressed.

Tape: A cassette-only BBC model B micro does not provide the *BUILD command. Exec files can only be written using a wordprocessor, and can't be auto-booted by pressing SHIFT–BREAK.

Net: An auto-boot file can only be put in the user root directory &. This is the directory reached with the *I AM command or with *DIR. Consult the network manager if in doubt.

Exec files such as !BOOT can also include commands to change the screen colour; this is explained a little later.

Special Effects With Printer Drivers

The Acorn Printer Driver Generator can be used to create extraordinary printer drivers that allow unusual effects to be used for special ViewSheet models. The essence of the technique is to *lie* to the generator. The special effects can be applied at three levels.

Whole Sheet Effects

The first type of special effect relates to the whole print-out.

One of the difficulties of using spreadsheets with many printers is that models may exceed the maximum width of the printer. Most printers used at home are only 80-columns wide. Using a dot-matrix printer, this restriction can be lifted by selecting a *condensed font* which usually gives up to 132 characters across the paper on an 80-column printer. The condensed font must be switched on before printing commences by sending the relevant control codes to the printer.

This can be achieved by specifying the necessary control codes when the generator requests the codes for printer initialisation. For example, the following answer will make an Epson FX80 or compatible printer set condensed mode when the printer driver initialises it:

```
Include printer initialisation? Y
Give code sequence for initialising the
printer:ESC "I" 1 15
```

The ESC" I "1 is the standard initialisation code for an FX80 or compatible printer. It allows the pound sign to be printed using control code 6. The extra code 15 sets condensed mode.

This initialisation code is sent by the printer driver to the printer every time printing begins. Other effects can be selected in a similar way: the printer could be set to near letter quality mode for example. If necessary, complex sequences of up to a maximum between seven and 21 separate codes can be entered into the initialisation routine (the actual maximum depends upon which codes are entered!). This allows ample scope for setting bizarre italic enlarged bold fonts if necessary.

Window Effects

Another type of special effect relates to the whole of a printer window. Highlight options one and two are normally attached to a printer window definition, and cause that particular window to be printed in bold face, or underlined. If for example, a combination of bold face and italics is needed instead, then press Y in reply to the generator's 'Include underlining?' question, but remember to give the codes for italics when the generator asks for the underline on and off codes. Answer the other questions in the normal way: press N in response to 'Include italics?'. When this special driver is used, highlight option one will print italics, even though the driver actually *thinks* it is switching on underlining.

Single Character Effects

Lying about the ASCII code that prints hash, dollar or pound signs, allows the printing of any single special character that the printer provides. Simply put a hash on the sheet. The driver will use the altered code, and the printer will print the special character. For example, a degree symbol can be printed on an Epson FX80 or compatible printer by specifying code 5.

Special Effects Without a Printer Driver

Another good way to set whole sheet effects is to use an exec file. An exec file can contain control codes, and these can be used to set special fonts on the printer. The advantage of using exec files is that the same files can be used with all languages, not just the VIEW family. Obviously, BASIC or Pascal don't use a VIEW family printer driver when printing, but an exec file that does something to the printer can be used in exactly the same way in BASIC, Pascal and ViewSheet.

Breaking the Code

Short exec files are usually written using *BUILD, but control codes can't normally be inserted into the file. The Master series micros have an enhanced version of *BUILD which allows control codes. |A is the equivalent of ASCII code 1 or CTRL–A. This coding may be familiar from function key definitions, which often contain |M to mimic RETURN (that is ASCII code 13). *BUILD doesn't allow editing of the file, which can be

194

a problem if the exec file doesn't work as expected. An alternative is to use the Master screen editor, EDIT. By using this, control codes can be inserted into a text file simply by pressing CTRL–A, for example. One slight difficulty with EDIT is inserting the ESCAPE code. As EDIT usually ignores this key, the way to do it is to use some unique word to represent ESCAPE in the text, such as XESCX, then at the end, type:

f5 'Global replace: XESCX/|['.

EDIT is only included with the Master 128. Producing special exec files containing any type of control codes can be made easier on all BBC micros. The DECODE program is the thing to use. It reads a normal text file written by VIEW or any other wordprocessor, and converts it into a special exec file. Control codes can be put in the text file in a special coded form (|M for RETURN and so on as previously mentioned), and they will be inserted into the exec file in decoded form.

Using DECODE

To use DECODE, type in the BASIC program in listing 1 at the end of the chapter, and save it; a suitable filename is DECOsrc. Then run the program. It will assemble and save a machine code file called DECODE.

This DECODE machine code program can now be used to read any text file and convert it to an exec file. If the text file contains control codes in coded form, these are decoded and put into the exec file. To do this, type in the following:

```
*DECODE <text-file> <exec-file>
```

giving the names of the old text file and the new special exec file. The DECODE program reads in the text and converts it into a new exec file.

If the exec file doesn't work as it should, then the original text file can simply be reloaded into a wordprocessor and edited. Finally, DECODE should be tried again. DECODE won't overwrite an existing file, so delete the non-working copy of the exec file before re-running DECODE. Once the exec file works properly, it can be used quite independently, but it is a good idea to keep the original text file somewhere in case additions need to be made later.

Using DECODE this way may destroy any other text or programs stored in the machine as it uses a chunk of normal memory to run in, and store data, so save any current work first before using DECODE. Altering lines 150, 160, 170 and 1600 as shown in the listing at the end of the chapter prevents this destruction, as the DECODE program then runs in memory normally used for cassette input, function keys buffers and user defined characters. These changes are not recommended with networked Master series machines, nor with 6502 second processors. DECODE does not work with the cassette filing system.

Condensed Mode Printing

DECODE can be used to create an exec file to make the printer print in condensed mode. Figure 13.1 is the original text that should be typed into VIEW or another wordprocessor and saved, perhaps in a file called 'CONDsrc'.

```
|B
|A|0
|C
*|| Condensed print on Epson printer|M
*|| (C) Graham Bell 1988 |M
```

Figure 13.1. Coded exec file for condensed print

DECODE can then be used to create the final exec file. Type:

*DECODE CONDsrc CONDENS

The new CONDENS file can be used to set condensed mode on all Epson-compatible dot-matrix printers. Before printing a model from ViewSheet which is over 80 characters wide, ensure the printer is connected and type in:

*EXEC CONDENS

The printer should now be ready to print up to 132 condensed characters across the paper. The normal ViewSheet PRINT command can be used, and the model will be printed in the condensed font.

Master Series: *EXEC CONDENS can be shortened to *CONDENS, because files can be executed automatically. This takes advantage of the facility whereby *RUN is the equivalent of *EXEC for any file that has its execution

address (the second address that appears after *INFO) set to &FFFFFFFF. This feature is also supported on the BBC model B by the Watford Electronics Disc Filing system.

CONDENS can even be used with a normal VIEW printer driver, providing the printer driver does not reset the printer and cancel the effect of CONDENS. The initialisation code to avoid for most dot-matrix printers is ESC "@".

Pound Printing Problems

The hash, dollar and pound signs are, as we have seen, a source of problems with printers. Figure 13.2 lists a text file that ensures that these problematic characters are printed correctly, even without a printer driver. It is for the popular Epson FX80, Kaga/Taxan or Canon printers, though it may work with others too. These printers have a redefinable character set held in RAM (often called the downloadable character set). The exec file selects the American character set so that hash and dollar print correctly, then redefines character 96 to be a £ shape rather than the grave accent (`) the printer usually uses.

```
|B
|A| [|AR|A|@
|A| [|A:|A|@|A|@|A|@
|A| [|A%|A|A|A|@
|A| [|A&|A|@|A£|A£
|A|!|J|A|R|A|@|A~|A|!|@|A|R
|A|!|@|A|R|A|!|@|AB|A|@|A|@
|C
*|| £ # $ on Epson FX80 printer|M
*|| (C) 1988 Graham Bell |M
```

Figure 13.2. Coded exec file for defining a pound sign

Type the text into VIEW or another wordprocessor, and save it : 'POUNsrc' or a suitable filename. Then use DECODE as previously described to create an exec file called 'POUND'. After that type in:

*EXEC POUND

which sets up the printer to use all the hash, dollar and pound characters correctly. Do this before printing your model. You can also use both POUND and CONDENS together.

VAV—N

The use of POUND need not be confined to ViewSheet. It is of equal use for printing BASIC listings, where hash and pound signs often get mixed up in keywords like BGET#. Simply execute the POUND file as above, before using the printer in the normal way.

Bespoke Exec Files

DECODE can be used to create special exec files to achieve other effects. First, find out how to do the job in BASIC, using VDU to send characters and ASCII codes to the printer. As an example, the BASIC program to set pica pitch and near letter quality (NLQ) mode on a Canon PW1080 or Kaga/Taxan 810 dot-matrix printer is:

```
10 VDU 2, 1, 27, 1, ASC "P"
20 VDU 1, 27, 1, ASC "(", 3
```

These printers don't select near letter quality in quite the same way as Epson printers. The equivalent on a truly Epson-compatible printer capable of NLQ printing is:

```
10 VDU 2, 1, 27, 1, ASC "P"
20 VDU 1, 27, 1, ASC "x", 1, 1, 3
```

Once the necessary VDU codes are known, their equivalents can be typed into a text file, using figure 13.3 as a guide. Therefore the two sets of BASIC lines above become respectively:

```
|B|A| [ |AP
|A| [ |A(|C
```

or alternatively:

```
|B|A| [ |AP
|A| [ |Ax|A|A|C
```

Any wordprocessor can be used to create this text file, which might be called 'NLQsrc'. In the wordprocessor file, there should be no formatting of the text, nor any embedded control codes.

ASCII code	ASCII code in hex	string in text file
0	00	|@
1 - 26	01 - 1A	|A - |Z
27	1B	|[
28	1C	|\
29	1D	|]
30	1E	|^
31	1F	|_
32	20	SPACE/
33 - 126	21 - 7E	! - ~
127	7F	|?
128	80 - 9F	|!|@
129 - 154	81 - 9F	|!|A - |!|Z
155	9B	|!|[
156	9C	|!|\
157	9D	|!|]
158	9E	|!|^
159	9F	|!|_
160	A0	|!SPACE/
161 - 254	A1 - FE	|!! - |!~
255	FF	|!|?

The ASCII codes between 33 and 126 are just the normal character set, including digits and upper and lower case letters. They are repeated between 161 and 254, prefixed by |!. The exceptions are the broken bar symbol |, which is coded as ||, and the quote mark " which is |". Finally, SPACE must be encoded as |SPACE, but only when it occurs at the start of a line. This coding, sometimes called GSREAD format, is the same as that used in function key definitions.

Figure 13.3. Control characters and GSREAD coded equivalents.

Then DECODE can be used on the text file to produce the final exec file, which might be called 'NLQ'.

    *EXEC NLQ

will then switch the printer into NLQ mode. Because of the limitations of most of these types of printer, the NLQ exec file should not be used with either CONDENS or POUND. One exception is the Citizen 120D printer, which allows condensed NLQ printing. Therefore the CONDENS and NLQ exec files can be used together.

There are four points to note about constructing new exec files. The printer must be switched on and off with I B (ASCII code 2) and I C (ASCII code 3). Codes which are sent with the printer off, have no effect whatsoever. All other control characters should be prefaced with I A so they are sent only to the printer. A RETURN character must be explicitly coded as I M because DECODE ignores the ends of lines in the original text file. Finally, notice that the text near the end of figures 13.1 and 13.2 starts with *I I; when decoded this becomes *I, which is the operating system's equivalent of a REM (reminder line). By using this technique, normal text can be incorporated into the exec file without any danger of it being accepted as a command. The text is useful because it serves as a reminder that the exec file has been used.

Setting a Page Layout

When you print a spreadsheet direct from ViewSheet, there is no way of setting a left margin. The sheet is usually printed flush against the left-hand edge of the paper. To prevent this, the paper has to be put into the printer so that it starts printing further across the sheet, or the model can be spooled using the VSXFER program explained in Chapter Six, and read into VIEW, where all the usual page layout options are available.

But many printers are intelligent enough to do the simple margin setting job themselves. On an Epson-compatible printer, the control code ESC" l" n set a left margin, of n character spaces. Therefore to set a 10 character left margin the printer must be sent ESC"l" 10. Note that this is in terms of characters, so if the printer is in condensed mode at the moment of setting, then the margin will naturally be narrower.

Similarly, ViewSheet does not divide the spreadsheet into pages. If the total number of lines in the printer windows exceeds the length of the paper in use, then printing continues inconveniently right across the page break. Again most printers are clever enough to deal with this themselves. ESC" C" n can be used to tell the printer how long the paper sheets are, and ESC "N" n to set a gap of n lines between the bottom line of one page and the first line of the next. These features are often called 'set form length' and 'set perforation skip' in the printer's manual.

So to set up a page format for A4 paper, with 70 lines per page, and one-inch (six-line) margins at left, top and bottom, set the paper in the printer at the place where the first line on the first page should be, and set the control codes:

```
ESC "C" 70
ESC "N" 12
ESC "1" 10
```

This translates into GSREAD coded form for DECODE as:

```
|A|[|AC|AF
|A|[|AN|A|L
|A|[|A1|A|J
```

To ensure that the printer is not in condensed mode before the left margin is set, the printer should be reset. This restores the printer to its default state, and is a more elegant way that switching off then on again. The codes for this are ESC"@", or:

```
|A|[|A@
```

All of these codes are combined in the text file in listing 2 at the end of the chapter. It is intended to define a good layout for A4 continuous stationery. The exec file, once decoded, does the following:

> Resets the printer to its switch-on state
> Sets a page length of 70 lines
> Sets a two-inch perforation skip
> Sets a one-inch left margin
> Defines the pound sign like the POUND program
> Sets condensed print mode

This layout allows about 115 characters across the page, rather than the normal 132 for condensed mode printing, because of the width taken up by the margins. However, only about 100 should be used to maintain a decent right margin and keep a balanced look to the pages.

Type in the text file, and save it, a name like 'A4Psrc' is suitable. Then use DECODE to produce the exec file containing the control codes: call that 'A4P'. Now whenever some printing must be done (without a printer driver), just set the paper in the printer to where the first line should appear, and type:

```
*EXEC A4P
```

and the printer will be instructed to set the page layout as described. The use of the A4P file is not confined to ViewSheet. BASIC listings can be printed out using this method, so they don't print over the perforations in the paper.

Using the intelligence built into the printer in this way is fine, but should not be mixed with the use of a printer driver. VIEW and ViewStore generally rely on having total control of the printer; they keep a count of the lines printed, and print blank lines to give margins at the head and foot of a page. They use spaces to maintain a left margin.

Changing Colour in !BOOT Files

If DECODE is used, an auto-exec file such as !BOOT can contain the control codes necessary to change the screen colours. The relevant keypresses are described in Chapter Five. These can be incorporated into a text file using, for example:

```
MODE 0|M
|S|0|4|0|0|0
|S|7|6|0|0|0
```

for light blue (cyan) text on a blue background. The text file can then be decoded to produce the !BOOT file.

The place in the !BOOT file to use these colour commands needs some thought. Obviously, they must be placed after any MODE command as above, because changing mode always resets the screen to its normal colours. But loading a set of window definitions into ViewSheet can also cause the mode to be changed, so the colour commands need to be placed after any LW too.

A Final Twist to Printing

What happens when the spreadsheet grows to more than the width of the printer, even in condensed mode? The maximum 136 character width of most printers is little more than half the full width of a ViewSheet model. The maximum overall width of the printer windows can be up to 255 characters. If there is a need to print out this full width, then there is only one course of action: print the sheet sideways!

The SIDEPRT program in listing 3 at the end of the chapter can be used to print even the largest model, in sections, down the length of many sheets of continuous stationery. It is compatible with most dot-matrix printers which have a graphics printing capability. More specifically those that support ESC "L" double density bit-image graphics and use ESC "A" to set the line-feed spacing. The most common exception is the Olivetti JP101 spark jet printer.

This program is a rewritten version of an original program by John Knight, published in *Acorn User* magazine, in April 1987.

SIDEPRT in Use

The SIDEPRT program does not print the spreadsheet directly; it can only print the contents of an ASCII file. So to print out any model, it has to be spooled to a file first. This can most easily be done using the ASCII file spooler described in Chapter Six. Once there is a version of the model in an ASCII text file on cassette or disc, then SIDEPRT can be used to print that file.

Type in the BASIC program, and save it using a name such as 'SIDEPRT'. This is a BASIC program, and so can't be used from within ViewSheet. Start up BASIC in the normal way, and type in:

```
CHAIN "SIDEPRT"
```

The program first asks for the filename of the spooled file to be printed out. Type this filename in, and press RETURN.

SIDEPRT reads in the first section of the file, displaying dots on the screen to confirm its progress. The text is then printed out sideways. Up to 38 lines of text are printed across the paper. The program can cope with the longest lines that can be generated by ViewSheet, up to 255

characters. If the file contains more than 38 lines, then the remaining text is read in after the first is printed, and this way a file of any size can be printed.

The program may be stopped by pressing ESCAPE at any time. The text file will be safely closed.

Inside SIDEPRT

If the printer is set to give automatic line-feed, then SIDEPRT double-spaces all the characters along the lines of text. This can be prevented, and the normal spacing restored, by changing line 80 to:

```
80 crlf$ = CHR$ 13
```

In order to print each character of the file sideways, the SIDEPRT program uses the definition of the character held in the micro and doesn't rely on the printer's character set. It reads this definition at the beginning of the assembler routine. This means that all characters, including pound and any special ones redefined using VDU 23, will be printed out just as they are shown on the screen. The characters are printed using the double-density bit-image graphics mode of the Epson-compatible printer.

The program can be modified for incompatible printers, providing they have some graphics printing capability. All the printer control codes used are defined at the start of the program. The codes in 'reset$' are used to reset the printer to its switch-on state. 'crlf$' sends the print head back to the extreme left and advances the paper one line. 'form$' feeds the paper to the top of the next sheet. The distance the paper moves up for each line is set by 'pitch$': the idea is to set the line pitch so that the characters along each line of text (reading sideways) are not too widely spaced. The last printer control string is used to put the printer into a bit-image graphics mode; the actual codes sent are ESC"L" m n, where m and n form the number of graphics dots to be printed. The constant 'dots%' is the maximum number of dots for that graphics mode. If 'dots%' is wildly different from 960, then the printed characters may look tall and spindly, or alternatively squat and wide, but in practice a wide variation is acceptable to the eye.

It should be noted that SIDEPRT can print any text sideways. It isn't limited to spooled ViewSheet models; any text file can be printed out, providing the line length doesn't exceed 255 characters.

Printing the Cell Contents

The ViewSheet PRINT command prints out the value or label in each cell that is visible in an active printer window. The formulae in the cells are not printed out, only the values of the formulae. ViewSheet has a separate command to print out the contents of cells, including formulae.

To print the cell contents for the entire sheet, type:

    P C

at the Command mode prompt. The type of output given is illustrated in figure 13.4, which relates to part of the BEACH pebble project model from Chapter Four. Each line consists of the name of a cell, followed by its contents. The contents include values (cell B14 for example), labels (cell D16) and, most importantly, the formulae (as in cell D14). Note that only occupied cells are listed, and the slots are printed out in the order in which they are calculated, starting at A1 and going left to right along each row in turn.

```
E12 D12^2
B13 10.1
D13 B13-AVERAGE(B6B14)
E13 D13^2
B14 9.0
D14 B14-AVERAGE(B6B14)
E14 D14^2
B15 -------
E15 -------
A16 mean
B16 AVERAGE(B6B14)
D16 variance
E16 AVERAGE(E6E14)
```

Figure 13.4. The contents of some of the cells in the 'BEACH' model

This type of listing of the cell contents can occasionally be very important in finding the mistakes in a model if things go wrong. For example, it makes it clear when a cell makes a forward reference. Of course, the BEACH model contains several forward references; cell D13

205

refers to AVERAGE(B6 B14), and cell B14 is forward. However, this isn't too bad, because the range B6 B14 contains only values, not formulae.

The PC command sends text via the printer driver, so the spooler programs ASCII or VSXFER can be used to create VIEW-readable files if necessary.

There is no SCREEN equivalent of the PC command. However, if the cell contents need only be shown on screen, or if a printer is not connected, then the following has the equivalent effect:

```
PRINTER
*FX 5 0
PC CTRL-N
```

This ensures that any characters sent to the printer by the PC command are lost, and not printed out. They are shown on the screen as normal. Pressing CTRL-N causes the screen to pause until SHIFT is pressed, after every screenful scrolls past.

When real printing must be resumed, the normal parallel printer should be selected, and the printer driver reloaded in the usual way:

```
*FX 5 1
PRINTER  FX4
```

Serial: Serial printers require *FX 5 2.

Net: Select the network printer with *FX 5 4.

Program Listings

```
 10 REM Exec file processor DECODE
 20 REM to copy GSREAD format files
 30 REM for BBC B/B+/M/C/2P/E
 40 REM (C) 1988 Graham Bell
 50 :
 60 osnewl = &FFE7
 70 osargs = &FFDA
 80 osbget = &FFD7
 90 osbput = &FFD4
100 osfind = &FFCE
110 gsread = &FFC5
120 gsinit = &FFC2
130 gspt = &F2
140 :
150 HIMEM = &2D00
160 code = &2D00
170 textln = &2F00
180 REM routine can be assembled at other suitable addresses
190 REM eg code = &A00 and textln = &C00 in stand alone BBC
    micro
200 REM note in line 1570 sum% <> &773A and delete line 150
210 :
220 argptr = &A8
230 :
240 FOR I% = 0 TO &1FF
250 I%?code = 0
260 NEXT
270 :
280 FOR pass = 0 TO 3 STEP 3
290 P% = code
300 [       OPT pass
310         LDA #1
320         LDX #argptr
330         LDY #0
340         JSR osargs
350         LDA argptr
360         STA gspt
370         LDA argptr + 1
380         STA gspt + 1
390 :
400         CLC
410         JSR gsinit
420         BEQ arger1
430         LDX #0
440         JSR cpynam
450 :
460         STX filtwo
470 :
480         CLC
```

```
490              JSR gsinit
500              BEQ arger2
510              JSR cpynam
520 :
530              LDA #64
540              LDX #filnam MOD 256
550              LDY #filnam DIV 256
560              JSR osfind
570              AND #255
580              BEQ filer1
590              STA ipfile
600 :
610              TXA
620              CLC
630              ADC filtwo
640              TAX
650              BCC open1
660              INY
670 .open1   LDA #64
680              JSR osfind
690              AND #255
700              BEQ openfi
710              STA opfile
720              JSR close
730 :
740 .filer2  BRK
750              OPT FNequb (131)
760              OPT FNequs ("Output file exists")
770 .arger1  BRK
780              OPT FNequb (128)
790              OPT FNequs ("No arguments")
800 .arger2  BRK
810              OPT FNequb (129)
820              OPT FNequs ("No output file")
830 .filer1  BRK
840              OPT FNequb (130)
850              OPT FNequs ("Input file not found")
860 .filer3  BRK
870              OPT FNequb (132)
880              OPT FNequs ("Can't open output file")
890              BRK
900 :
910 .openfi  LDA #128
920              JSR osfind
930              AND #255
940              BEQ filer3
950              STA opfile
960 :
970 .getln   LDX #0
980              LDY ipfile
990 .getln1  JSR osbget
1000             BCC getln2
1010             LDA #13
```

```
1020 .getln2 ROR eoflag
1030         STA textln,X
1040         INX
1050         CMP #13
1060         BNE getln1
1070 :
1080         LDA #textln MOD 256
1090         STA gspt
1100         LDA #textln DIV 256
1110         STA gspt + 1
1120         LDY #0
1130         SEC
1140         JSR gsinit
1150 :
1160 .putln  JSR gsread
1170         STY temp
1180         LDY opfile
1190         BCS putln1
1200         JSR osbput
1210         LDY temp
1220         BNE putln
1230 :
1240 .putln1 BIT eoflag
1250         BPL getln
1260 :
1270 .close  LDA #0
1280         LDY ipfile
1290         JSR osfind
1300         LDY opfile
1310         JSR osfind
1320         JMP osnewl
1330 :
1340 .cpynam JSR gsread
1350         BCC cpy1
1360         LDA #13
1370 .cpy1   STA filnam,X
1380         INX
1390         BCC cpynam
1400         RTS
1410 :
1420 .eoflag OPT FNequb (0)
1430 .filtwo OPT FNequb (0)
1440 .ipfile OPT FNequb (0)
1450 .opfile OPT FNequb (0)
1460 .temp   OPT FNequb (0)
1470 :
1480 .filnam
1490 :
1500 ]
1510 NEXT
1520 :
1530 sum% = 0
1540 FOR I% = 0 TO &1FF
```

209

```
1550 sum% = sum% + I%?code
1560 NEXT
1570 IF sum% <> &79F6 THEN PRINT "Assembler error - please
     check listing": END
1580 :
1590 PROCoscli ("SAVE DECODE " + STR$~code + " +200" +
     STRING$(2, " " + STR$~(&FFFF0000 OR code)))
1600 END
1610 :
1620 :
1630 :
1640 DEF FNequb (byte%)
1650 ?P% = byte%
1660 P% = P% + 1
1670 = pass
1680 :
1690 DEF FNequs (string$)
1700 $P% = string$
1710 P% = P% + LEN string$
1720 = pass
1730 :
1740 DEF PROCoscli (string$)
1750 LOCAL X%, Y%
1760 DIM X% &FF
1770 Y% = X% DIV 256
1780 $X% = string$
1790 CALL &FFF7
1800 ENDPROC
```

Listing 13.1. GSREAD format file decoder

```
|B
|A|[|A@
|A|[|AC|AF
|A|[|AN|A|L
|A|[|Al|A|J
|A|[|AR|A|@
|A|[|A:|A|@|A|@|A|@
|A|[|A%|A|A|A|@
|A|[|A&|A|@|A£|A£
|A|!|J|A|R|A|@|A~|A|!|@|A|R
|A|!|@|A|R|A|!|@|AB|A|@|A|@
|A|O
|C
*|| A4 page layout|M
*|| (C) 1988 Graham Bell|M
```

Listing 13.2 Coded exec file to set up A4 page layout.

```
 10 REM SIDEPRT sideways printer
 20 REM for ViewSheet spreadsheets
 30 REM for BBC B/B+/M/C/E
 40 REM (C) 1988 Graham Bell
 50 :
 60 REM printer control strings
 70 reset$ = CHR$ 27 + "@"
 80 crlf$ = CHR$ 13 + CHR$ 10
 90 form$ = CHR$ 12
100 pitch$ = CHR$ 27 + "A" + CHR$ 8
110 graphics$ = CHR$ 27 + "L"
120 dots% = 960
130 :
140 ON ERROR PROCerror (0)
150 PRINT "SIDEPRT"
160 :
170 file% = FNopen (FNname ("Spool file"), "R")
180 :
190 MODE 7
200 maxlines% = dots%/20
210 DIM buffer% 255 * maxlines%
220 DIM start%(maxlines%)
230 matrix% = FNassemble
240 :
250 PROCprinter (reset$)
260 PROCprinter (pitch$)
270 REPEAT
280 lines% = FNread_lines (maxlines%,file%)
290 chars% = FNmax_length (lines%)
300 dots$ = CHR$ ((lines% * 20) MOD 256) + CHR$ ((lines% * 20)
    DIV 256)
310 FOR char% = 0 TO chars% - 1
320 PROCprinter (graphics$ + dots$)
330 FOR line% = lines% - 1 TO 0 STEP -1
340 A% = FNget_char (line%, char%)
350 CALL matrix%
360 NEXT
370 PROCprinter (crlf$)
380 NEXT
390 PROCprinter (form$)
400 UNTIL EOF# file%
410 CLOSE# file%
420 PROCprinter (reset$)
430 END
440 :
450 :
460 :
470 DEF FNread_lines (maxlines%, file%)
480 LOCAL line%, ptr%, byte%
490 FOR line% = 0 TO maxlines%
500 start%(line%) = 0
510 NEXT
520 line% = 0
```

```
 530 ptr% = 0
 540 REPEAT
 550 byte% = BGET# file%
 560 IF (byte% = &0D) OR (byte% > &1F) THEN buffer%?ptr% =
     byte% : ptr% = ptr% + 1
 570 IF byte% = &0D THEN line% = line% + 1 : start%(line%) =
     ptr% : PRINT ".";
 580 UNTIL (line% = maxlines%) OR (EOF# file%)
 590 IF POS <> 0 THEN PRINT
 600 = line%
 610 :
 620 DEF FNmax_length (maxlines%)
 630 LOCAL line%, length%, max%
 640 max% = 0
 650 FOR line% = 0 TO maxlines% - 1
 660 length% = start%(line% + 1) - start%(line%) - 1
 670 IF length% > max% THEN max% = length%
 680 NEXT
 690 = max%
 700 :
 710 DEF FNget_char (line%, char%)
 720 IF (char% >= (start%(line% + 1) - start%(line%) - 1)) THEN
     = 32 ELSE = buffer%?(start%(line%) + char%)
 730 :
 740 DEF FNassemble
 750 DIM addr 60, block 8
 760 osword = &FFF1
 770 oswrch = &FFEE
 780 FOR pass = 0 TO 2 STEP 2
 790 P% = addr
 800 [         OPT pass
 810           STA block
 820           LDA #2
 830           JSR oswrch
 840           LDA #10
 850           LDX #block MOD 256
 860           LDY #block DIV 256
 870           JSR osword
 880 :
 890           LDX #7
 900 .loop1    LDA block + 1,X
 910           JSR print
 920           JSR print
 930           DEX
 940           BPL loop1
 950           LDX #3
 960           LDA #0
 970 .loop2    JSR print
 980           DEX
 990           BPL loop2
1000 :
1010           LDA #3
1020           JMP oswrch
```

```
1030 :
1040 .print   PHA
1050          LDA #1
1060          JSR oswrch
1070          PLA
1080          JMP oswrch
1090 .end     BRK
1100 ]
1110 NEXT pass
1120 IF end-addr <>&83 THEN PROCerror(2)
1130 =addr
1140 :
1150 REM write string to printer only
1160 DEF PROCprinter (string$)
1170 LOCAL char%
1180 VDU 2
1190 FOR char% = 1 TO LEN string$
1200 VDU 1, ASC (MID$(string$, char%))
1210 NEXT
1220 VDU 3
1230 ENDPROC
1240 :
1250 DEF FNname (p$)
1260 LOCAL n$
1270 PRINT ' p$;
1280 INPUT "? " n$
1290 REM delete trailing spaces
1300 REPEAT
1310 IF RIGHT$(n$, 1) = " " THEN n$ = LEFT$(n$, LEN n$ - 1)
1320 UNTIL RIGHT$(n$, 1) <> " "
1330 = n$
1340 :
1350 DEF FNopen (n$, c$)
1360 LOCAL h%
1370 c$ = LEFT$(c$, 1)
1380 PRINT "Opening "; n$
1390 IF c$ = "R" THEN h% = OPENIN n$
1400 IF c$ = "U" THEN h% = OPENUP n$
1410 IF c$ = "W" THEN h% = OPENOUT n$
1420 IF h% = 0 THEN PROCerror (1)
1430 = h%
1440 :
1450 DEF PROCerror (number%)
1460 CLOSE#0
1470 IF number% = 0 THEN REPORT : PRINT
1480 IF number% = 1 THEN PRINT "Can't find file"
1490 IF number% = 2 THEN PRINT "Assembler error"
1500 END
```

Listing 3. Prints ASCII files sideways.

14 : Special Spreadsheet
Functions
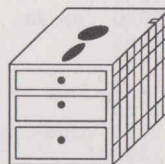
ROW and COL

The ROW and COL functions, introduced in Chapter Three, are often used simply to turn the location of a formula into a number. The multiplication table example given in Chapter Three uses this property, but there is another important application of these two functions.

Commonly, a spreadsheet is used to maintain a list of information, a sort of primitive database. Additional items can be added to the list by inserting whole rows or columns of cells. An example of this approach is the household budget model 'HOUSE' developed in Chapter Four, with the list of household expenses. But how can the number of items in the list be found? The initial number in this model was five items, but more could be added. Because inserting a row or column adjusts all the formulae intelligently, functions like AVERAGE and MIN continue to work properly, but if the size of the list is needed for any other calculation, then this must be found in another way.

The answer is given by ROW. Obviously, if ROW were used at the head and foot of the list, then the length of the list is the difference between the two. One possible layout is shown in figure 14.1. The cells A3 and A9 both contain the formula ROW, and the length of the list (seven items) can be found at any time by using A9 - A3 + 1 as in cell C11.

```
V A  SLOT=C11
CONTENTS=A9-A3+1

0       ............A............B............C
......1
......2
......3              3 Rent
......4                Rates
......5                Water
......6                Gas
......7                Phone
......8                Electricity
......9              9 Petrol
.....10
.....11                No of Items ▷▬▬▬▬▬▬▬
```

Figure 14.1. Finding the length of a list.

214

This technique could also be useful in finding the number of pebbles in the list on the Pebble Project model from Chapter Four.

Conditional Functions

There are three special spreadsheet functions supported by ViewSheet that may be described as conditional. The first and most familiar to most people is the IF function; the other two are CHOOSE and LOOKUP.

IF Only...

The IF function is used to select one of two possible outcomes. In English, you might say:

```
IF the egg floats THEN it's bad OTHERWISE it's fresh
```

The selected outcome depends on whether the condition, 'the egg floats', is true or false. ViewSheet has a very similar structure:

```
IF (<condition>, <outcome-1>, <outcome-2>)
```

If the condition is true, then outcome-1 is selected. So the following formula has the value 10 when A1 is greater than A2, or 0 otherwise:

```
IF (A1 > A2, 10, 0)
```

The IF functions can also be nested, so that for example outcome-1 could then select between two further sub-outcomes. This nesting is important because ViewSheet has no AND or OR operators like BASIC. Conditions cannot be combined. Consider the following formula:

```
IF (cond-1, IF (cond-2, 10, 20), IF (cond-2, 20, 30))
```

This can be used to test for any combination of conditions one and two. If both are true (this equates with AND), then the value of the formula is 10; if either one or the other are true then the value is 20, and if both are false, then the formula value is 30.

Frequency Tables

An interesting use of the IF function is in the building of frequency distributions. As an example, the Pebble Project model BEACH can be used. In this, a list of beach pebble sizes is given. The objective is to

divide them into groups or classes, and show how many are between six and eight centimetres long, how many between eight and 10, and so on. This is a frequency table; it shows the number of pebbles in each size group. The frequency distribution shows how the number in each group varies. Some groups are large (there might be 15 pebbles between six and eight centimetres in the sample), others are very small (only two pebbles between 16 and 18 cm).

Figure 14.2 shows the BEACH model loaded in, with the display divided into two windows (note the gap between C and G along the top border). The first window shows cells B1 to C16 as they were originally. The new window shows the area G1 J16: this will become an occurrence table, from which the frequency distribution can be derived.

```
VA   SLOT=G6
CONTENTS=IF(B6>=G2,IF(B6<G4,1,0),0)
```

0	.........B.........C		......G......H......I......J			
......1	BEACH	PEBBLE				
......2	-----	------	6	8	10	12
......3	length		to	to	to	to
......4	cm		8	10	12	14
......5						
......6	11.1	███████	0	1	0	
......7	12.3					
......8	7.9					
......9	10.4					
.....10	10.8					
.....11	9.6					
.....12	10.2					
.....13	10.1					
.....14	9.0					
.....15	-------					
.....16	10.2					

Figure 14.2. Setting up an occurrence table for the pebble data.

The information in rows two and four of the occurrence table are to be used as the lower and upper limits of each frequency class. They are not labels, but are put in the form shown so that the limits (six in cell G2 for example) can be refered to in other formulae. The first four formulae are shown in row six of the table.

The formula in cell G6 is constructed so that it gives the value one only if the pebble length in cell B6 is both greater than the class minimum and less than the class maximum, otherwise the value should be zero:

```
IF (B6 >= G2, IF (B6 < G4, 1, 0), 0)
```

This is equivalent to a 'tick' in the box if the pebble belongs to the group represented by that column.

The formula can be repeated across the whole of the occurrence table, taking care over the absolute and relative references. As a check, the formula in cell J14 should be:

```
IF (B14 >= J2, IF (B14 < J4, 1, 0), 0)
```

Notice that the >= conditional operator is used for the lower limit, < for the upper limit. This is to avoid problems when a pebble length value lies exactly on the boundary between two classes. By arranging the conditional operators like this, each value can satisfy the conditions for only one class.

When the table is complete, there should be only a single non-zero value in each of the rows, marking the class within which the pebble length falls (that is, one tick per pebble). Each of the class columns can be totalled to show the number of pebbles from the sample in each class (the number of ticks for each group). Figure 14.3 shows the model with the occurrence table completed, with the frequencies in each class totalled in row 16.

```
IA   SLOT=B27
CONTENTS=6 to 8
Wi  TopL  BotR  Pos  Cw  Bw  Fmt  Opt
2   B25   C30   B0   9   7   FRM  TC
0        .........B.........C.......G......H......I......J
......1       BEACH    PEBBLE
......2       ------   -------        6      8      10     12
......3       length                  to     to     to     to
......4         cm                    8      10     12     14
......5
......6        11.1                    0      0      1      0
......7        12.3                    0      0      0      1
......8         7.9                    1      0      0      0
......9        10.4                    0      0      1      0
.....10        10.8                    0      0      1      0
.....11         9.6                    0      1      0      0
.....12        10.2                    0      0      1      0
.....13        10.1                    0      0      1      0
.....14         9.0                    0      1      0      0
.....15       -------               ------ ------ ------ ------
.....16        10.2                    1      2      5      1

.....25       FREQ.  HISTOGRAM
.....26       -----  ---------
.....27 > 6 to  8 *
.....28    8 to 10 **
.....29   10 to 12 *****
.....30   12 to 14 *
```

Figure 14.3. Frequency distribution bar chart for the BEACH model.

To construct tables like these, a short-cut can be used in ViewSheet. Each cell can be filled with a formula meaning (length >= minimum) * (value < maximum). This works because a false condition has the numerical value zero, a true condition the value one. The formula multiplies the two conditions together, and only if they are both true is the overall formula value one. However, this approach is less likely to work with other spreadsheets, which often assign different numerical values to true and false.

Charting Success

Figure 14.3 also shows a new third window and its definition. The new window shows a histogram or bar chart of the frequency distribution of pebble lengths. All the values in any window can be displayed as a bar chart merely by placing C in the Opt part of the window definition. The bars are rows of asterisks, the length of the row controlled by the value in the cell. Negative values are shown by rows of exclamation marks. Printer windows can be converted to bar charts in the same way. Unfortunately, all bar charts drawn this way have to be horizontal, because each bar is printed within a single cell. To make proper graphs, ViewPlot or ViewChart must be used.

To build the bar chart in figure 14.3, the cells C27 to C30 each refer to one of the column totals in row 16. So the formula in cell C29 is simply I16, and the value of the formula is five, so the bar is five asterisks long.

It is always worth checking that the column width of the chart window is large enough for the bars to be shown in full. With more pebble measurements than are used in the example, the number in a single class might exceed nine. If a value exceeds the width of the column, the bar is truncated. The maximum value that can be displayed properly is equal to the column width. Normally the cure is to increase the column width so all of the bar can be displayed.

If the column width can't be increased, then the scale of the chart can be reduced. Normally, the scale is one to one: a value of five means five asterisks. But the cell can have a scale factor, so cell C27 above could contain the formula I16 * 0.5. This would half the scale. Alternatively, I16 * 2 could be used to increase the scale.

All the values for a whole window are converted to bars. For this reason, charts are normally allocated a window of their own. Figure

14.3 also shows that a bar chart window may contain labels as normal; they are not affected.

After construction of a bar chart like this, the window definitions should be saved with the SW command. There should then be two separate sets of window definitions for the same model; each could be loaded to study the data in a different way, one giving a statistical perspective, the other a graphical view of the same data.

Choosing Between Possible Values

The second conditional function is CHOOSE. This can be used to select one numerical item from a list. The general form of the syntax is CHOOSE (<item>, <list>). The list is a series of values separated by commas, so the function below:

```
CHOOSE (3, 10, 9, 8, 7, 6)
```

has the value 8, since 8 is the third item out of five in the list.

The list of values may include ordinary numbers, cell references, other functions and ranges. If a range is used, then it has the effect of including the sum of the cells in the range as a single item in the list. Therefore the following:

```
CHOOSE (3, 10, C7 C11, C13)
```

gives the value of whatever is in cell C13, and not the value of cell C8 as might be expected.

This may not seem all that useful: why not just have a cell reference to C13? CHOOSE is used mainly because the item to be selected can itself depend on another calculation. This function returns the number of days in a month selected by the value in cell A3:

```
CHOOSE (A3,31,28,31,30,31,30,31,31,30,31,30,31)
```

So if cell A3 contains four, the CHOOSE formula gives the number of days in April, the fourth month.

Because CHOOSE can't choose item number 4.8 from the list, so the item number in cell A3 in the above example must be a positive whole number. If a negative number or zero is used, then '?Error' is displayed in the CHOOSE formula slot. If a real number is used, then this is

truncated; 4.8 selects the fourth list item, 0.5 selects the zero item and so gives an error.

Go and Look it Up!

The third ViewSheet conditional function is LOOKUP. The syntax is:

```
LOOKUP (<value>, <compare-range>, <result-range>)
```

This function is generally used to look up a value in a table. It can sometimes be used in the same way as CHOOSE. For example if a table were set up like that in figure 14.4, then the formula LOOKUP (A3, A10 A21, B10 B21) would give the number of days in the month according to the value in cell A3. LOOKUP works by comparing the value (in cell A3) with each of the values in the first range A10 A21. When it finds one that is the same as A3, then it returns the corresponding value in the second range B10 B21.

```
 0       .......A.......B
...10 >        1        31
...11          2        28
...12          3        31
...13          4        30
...14          5        31
...15          6        30
...16          7        31
...17          8        31
...18          9        30
...19         10        31
...20         11        30
...21         12        31
```

Figure 14.4 Length of each month in days.

But LOOKUP is different from CHOOSE: if A3 in the above examples contains the value 4.8, CHOOSE returns the number of days in April, but LOOKUP gives an error because no cell in the LOOKUP range contains 4.8. It has to be an exact match for LOOKUP to work.

LOOKUP is usually used for processes like finding prices in price lists. If each type of item has a unique code number (like bar codes on supermarket goods), then a price list can be set up. It should contain the product code numbers in the LOOKUP range and the corresponding prices in the result range.

Conditional Weaknesses

The major problem with all the conditional functions is that they cannot deal with labels at all. So with the price list example, the product codes may not contain any letters or they will be treated as labels. The coding system used in Habitat shops (whereby 718300 is a rolling pin) is acceptable, but the Acorn item code system (SBB07 is a ViewSheet ROM) is not.

Many other spreadsheets (though not on the BBC micro) have a LOOKUP function that acts more intelligently, so an exact match is unnecessary. If the LOOKUP range is in order (lowest numbers first), then LOOKUP finds the largest number in the LOOKUP range that is not greater than the value given. For example, this method can be used to find the top (or marginal) rate of income tax paid according to a person's salary. In this case the LOOKUP range contains the thresholds at which the tax rate changes: the point where basic rate tax is paid for the first time and the points where enhanced rates start. The result range of course contains the percentage rates themselves.

Trigonometry and Logarithms

ViewSheet provides a range of the normal trigonometric functions; SIN (a), COS (a) and TAN (a) and their inverses ASN (n), ACS (n) and ATN (n).

These trigonometric functions expect the angle a in radians, or return the angle related to the value n in radians. One radian is about 57 degrees, but the following two functions can be used to convert easily from degrees to radians and back again.

> convert angle from degrees to radians   RAD (a)
> convert back from radians to degrees   DEG (a)

So if a is an angle in degrees, its cosine is given by the formula COS (RAD (a)); first convert a to radians, then take the cosine. Conversely, to convert a tangent n to an angle in degrees, use DEG (ATN (n)); take the arctangent and then convert to degrees.

Another useful function is PI, the value of π. This requires no brackets, as it has no argument. It always gives the value 3.141592653.

LN (n) is the natural (base e) logarithm of n. The natural anti-logarithm is given by EXP (n). LOG (n) is the (base 10) logarithm of n. There is no explicit anti-logarithm function, but the formula 10 ^ n can be used instead.

To complete the list of ViewSheet functions, ABS (n) is the magnitude, or absolute size, of the argument n. ABS ignores the minus sign if n is negative, so:

```
ABS  (-5.1)  = 5.1
ABS  (7.5)   = 7.5
```

INT (n) strips off the fractional part of a number and leaves only the whole number, so that

```
INT  (-5.1)  = -5
INT  (7.5)   =  7
```

For negative numbers, this is different from the BASIC version of INT, which returns the next lower integer. In BASIC:

```
INT  (7.5)   =  7
INT  (-5.1)  = -6
```

SGN (n) gives one of three values, 1 if n is a positive number, 0 if n itself is 0, or –1 if n is negative. Thus SGN (–5.1) = –1 and SGN (7.5) = 1.

SQR (n) is the square root of n: n must be positive, or an error will occur.

15 : Spreadsheet Integration
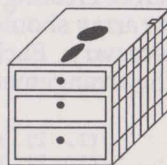
ViewSheet has a special facility for linking several spreadsheet models together. Final figures in one model may be needed to form part of a second model and at other times, the figures from several models may need to be collated together in an overall spreadsheet. These figures can be transferred automatically from one model to another.

The method does not work with the Cassette Filing System, because it makes use of random-access files, but is fully compatible with the DFS, ADFS or the Econet network.

The general procedure is as follows. First, create a file on a disc to hold the figures to be transferred. Then on the first model, write the figures into the file. Now the second model can read the figures directly from the file instead of copying them across by hand. The special files are called *link files*, or *array files*.

Creating a Link File

The ViewSheet command to create a special file for this purpose is:

```
CREATE <number> <max-x> <max-y>
```

where <number> controls the name of the link file. The files must always be called 'V.VS<number>', so if <number> is one, the file is called 'V.VS1'. <number> can be up to 255.

The size of the link file is controlled by <max-x> and <max-y>. The file is arranged in rows and columns, but these needn't be the same as the rows and columns in the original spreadsheet. <max-x> is the number of columns in the array, and <max-y> is the number of rows. So (<max-x>, <max-y>) are the co-ordinates of the bottom right-hand cell in the array.

Before creating a link file in this way, the arrangement of values within the array should be worked out, so that the file is no bigger than necessary. Each cell within the array is known by its (x, y) co-ordinates. The arrangement of cells within a four by five link file is as below:

```
(1, 1)  (2, 1)  (3, 1)  (4, 1)
(1, 2)  (2, 2)  (3, 2)  (4, 2)
(1, 3)  (2, 3)  (3, 3)  (4, 3)
(1, 4)  (2, 4)  (3, 4)  (4, 4)
(1, 5)  (2, 5)  (3, 5)  (4, 5)
```

Each value to be transferred should be allocated to a certain cell within the file. Once this is done, the maximum (x, y) co-ordinates can be used in the CREATE command. Perhaps a four by five link file could be used to transfer the sales totals of five departments, for the four quarters of the financial year.

To create a file with maximum co-ordinates (4, 5) as shown, 20 cells in a four by five array, type in:

```
CREATE 1 4 5
```

at the Command mode prompt. The disc drive whirrs and a link file called 'V.VS1' is created, with room in it for 20 values.

ADFS, Net The directory V must be created beforehand,using the *CDIR command.

Writing and Reading Link Files

ViewSheet has a special function to write to a link file. For example, to put the value 50 into the file V.VS1, at position (4, 2) in the array:

```
WRITE (1, 4, 2, 50)
```

This is a function; it has to be put into a cell on the sheet. The value displayed in the slot is the value written to the file. In fact, it is usual to write the value of another cell or function into the file, rather than a number like 50, but any cell reference, formulae or value can be written. Remember, however, that the link file may only contain numbers; it is the value of a formula that is written, not the formula itself. For this reason, labels can't be used in link files either.

Reading a value back from a link file is accomplished in a similar manner. The READ function is used - so to read the value written above into a cell, use:

```
READ (1, 4, 2)
```

Viewheet can read a value before it has been written, because the CREATE command automatically fills the link file with zeros. An unwritten value always reads as zero.

In both cases, reading and writing, the disc drive may be accessed while the spreadsheet is recalculating. With a small file, this is a very quick operation, but as the link file grows, the time taken for reading and writing the file becomes longer. If the time taken gets too long, switch the recalculation mode to manual by pressing SHIFT-f0. An 'M' is displayed in the status area. In manual mode, recalculation of the whole sheet then takes place only when SHIFT-f7 is pressed, so the time taken is reduced to a minimum.

Link files with up to 41 entries (a four by 10 array for example) can be held in memory by the Disc Filing System, so reading from and writing to them is very quick. Any newly written values are only stored permanently on the disc when you press ESCAPE to leave the Sheet mode. Since the link file is open while in Sheet mode, this means BREAK should not be pressed.

Transferring Values Using a Link File

The commonest use for a link file is to transfer values from one model to another. You might want to do this because the two models have developed separately, or perhaps because each is too large to be combined with the other in the available memory.

As an example, let's consider the GADGET model from Chapter Four. This represents a manufacturing account for a ficticious small company. The final cell contains the overall cost of manufacturing items to sell. This cost is transferred to the trading account which also contains the costs of selling the items produced, and also to the ultimate profit and loss account.

The transfer between the models can be done automatically by creating an array file to contain the values. Each model can read the relevant value from the file, where it was written by the previous model. The transfer only involves a single value between perhaps three models, but it is best to leave some spare space in the array file, in case the requirements of the model change. With the GADGET file, a 10-element array seems adequate. To create this file, with the name 'V.VS2', use the following command:

```
CREATE 2 5 2
```

Load in the GADGET file, and switch to the Sheet mode. The final accounted cost to be transferred to the next spreadsheet is calculated in cell C37. The formula in C37 is the sum of C33 and C34. This can be replaced by the function:

```
WRITE (2, 1, 1, C33 C34)
```

The effect of this is that the final cost will be put in cell (1, 1) of the array file V.VS2. Note that the sheet still displays the same number as before. The displayed value of a cell containing a WRITE function is the value that is written to the file, in this case the sum of C33 and C34.

When the second model, the trading account, is constructed, it can read the value from the file at any time by putting the function:

```
READ (2, 1, 1)
```

in the relevant cell.

Consolidation

Another application of array files might loosely be called *consolidation*. This is the gathering together of results from several spreadsheets into one overall model. To illustrate this let's look at the Pebble Project models. In this classroom project, the variations in pebble size from a beach were investigated. When back in the school,samples from different parts of the beach, for example the shingle zone, the gravel and sand, could be investigated by different groups of children. Once data has been collected and analysed for each zone of the beach singly, the variations *between* different parts of the beach could then be studied.

One way to do this would be to put all the data onto a single huge spreadsheet. However, this is not the best method for two reasons. First, it prevents groups working separately on their own portions of the data. Second, with a large amount of data, the BBC micro may run out of memory to store the increasingly large model.

A good solution is to have each group working on a separate model. Each model can then be made to write its results into a link file. The link file can be read by an new model, which can be used to analyse the overall picture. The way for each group to build a similar model was covered in Chapter Two, using a *mask file*.

The mask file could be amended so that each model built puts its results into the link file. This can be done by making the following changes to the basic BEACH model:

```
A1      GROUP
B16     WRITE (3, 1, A2, AVERAGE (B6 B14)
E18     WRITE (3, 2, A2, E17 / B16 * 100)
```

This uses a link file called 'V.VS3'; the file should be created with the command:

```
CREATE 3 4 <groups>
```

where <groups> is the number of separate sub-models. Thus the array has four elements for each group. Only two of these elements are used by the WRITE functions above, the first column for the mean and the second column for the co-efficient of variation. Each group should be given a unique number, to be put in cell A2. The group number is used to control which row of the link file array is used by each group's model.

As each group loads the mask file and enters data into the model, the results are automatically written to the correct place in the link file. This works very well on a network, but you can do it just as well with a single classroom micro, with all the files being kept on a floppy disc.

When each group has completed its work, and the data has been written into the link file, the data can be read into an overall model. This could show bar graphs of the average pebble length in each sample, together with the co-efficient of variation. The graphs might well illustrate how the size decreases down the beach, from shingle to sand, while the co-

efficient remains nearly constant. This would mean that the pebbles and sand, though different sizes, are equally well sorted.
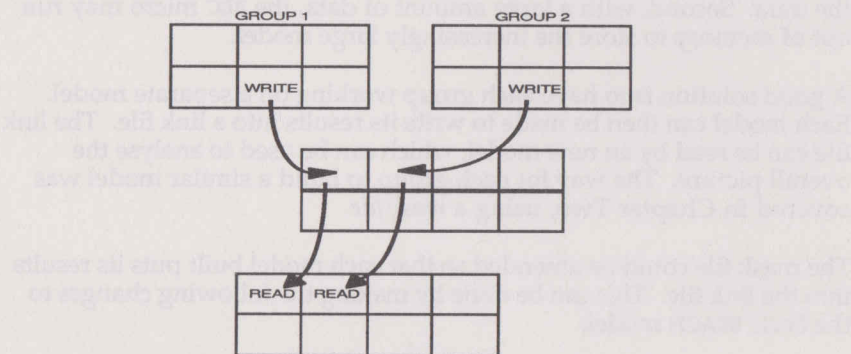
Figure 15.1. Information flow through a link file.

One drawback of the link file is that it is only updated when a model is recalculated. If several models are linked together, then a change made to the first file does not ripple through to the last model in the chain automatically. Each model 'downstream' in the information flow must be loaded and recalculated. A recalculation can be forced, however, by pressing SHIFT-f7.

Writing to Link Files from BASIC

The internal structure of a link file is compatible with BBC BASIC data files, so information can be put into link files from other programs written in BASIC.

The listing at the end of this chapter gives a set of procedures and functions to allow reading and writing of link files from BASIC. The function to open a link file must be used first, and this returns the file handle and the maximum co-ordinates of the link file, in a three byte array, at 'handle%'. PROCwrite and FNread work in the same way as their equivalents in ViewSheet, and use the data held at handle%. Various errors can occur if the file specified is not a link file, or if the X and Y co-ordinates exceed the size of the array in the file. Examples of how to use PROCopen, PROCwrite and FNread, as well as how to close the link file after use are given in the listing.

A BASIC program like this could be used to swap information between specially-written data gathering programs and ViewSheet models. Note that the BASIC program doesn't create the link file, this still has to be done by ViewSheet. With BASIC 1, replace OPENUP in line 430 with OPENIN.

ViewStore and Link Files

ViewStore has two methods of putting data into link files, so that numerical material from a database can be transferred to ViewStore.

The first is a special LINK utility. Run this like any other utility; load the database, and making sure there is plenty of memory free, type:

UTILITY LINK

The usual 'Use select file?' prompt appears. With the LINK utility, a selection will almost certainly be necessary. One row of the link file is created for each record in the database, and the largest size a link file can be is 255 rows. So make sure the database has fewer than 255 records, or select carefully which records are to be used by the utility.

The next set of prompts are for the names of the fields to be put into the link file. As the link file can only contain number, these must be numeric fields. As usual, wildcards can be used in the fieldnames, and even the field numbers can be used instead. Pressing RETURN on its own ends the list of fields. Each field forms one column of the link file. The link file array might look like this:

```
(f1, r1) (f2, r1) (f3, r1) (f4, r1)
(f1, r2) (f2, r2) (f3, r2) (f4, r2)
(f1, r3) (f2, r3) (f3, r3) (f4, r3)
(f1, r4) (f2, r4) (f3, r4) (f4, r4)
(f1, r5) (f2, r5) (f3, r5) (f4, r5)
```

for a five-record database, with four numeric fields used from each record.

Finally, ViewStore asks for a name for the link file. The CREATE command does not need to be used as the LINK utility does this automatically. As far as the LINK utility is concerned, the link file name

VAV—P

does not need to be of the form 'v.vs<number>', but the file must be called that so it can be read by ViewSheet. So type in a filename like:

```
Output filename? V.VS7
```

When the link file is complete, it will usually be necessary to copy it onto the disc containing the relevant spreadsheet.

ADFS, Net :   The filename can be much longer, and the file can be created directly in the ViewSheet directory where it will eventually get read:

```
&.VSHEET.V.VS7
```

would create the file 'VS7' in the VSHEET .V directory.

Figure 15.2 shows the creation of a single column link file with 24 rows to carry the invoice amounts from the CREDIT example database from Chapter Seven.

```
=>UTILITY LINK
LINK
Use select file (N,Y)? Y
Field 1? AMOUNT
Field 2?
1 field
Output file name? V.VS8
Creating file........................
File is 1 by 24
=>
```

Figure 15.2. Creating a link file from ViewStore.

In ViewSheet, a cell could contain a formula like:

```
READ (8, 1, 5)
```

to read the invoice amount for the fifth company in the ViewStore database, or at least the fifth in the selection file. That is column one, row five of the array. Any blank fields in the database get written into the link file as zeros.

The second method of producing a link file from a ViewStore database is via the REPORT utility described in Chapter 12. This offers the prompt:

```
Send totals to linking file (N,Y)?
```

Answer 'Y' (for yes) to create a link file. The idea behind this is that any register values used on subtotal or total lines in the report are also written to a linking file. Only subtotal and total registers can be sent to the link file. If the total line in the report definition file were:

```
T.Half1....................Half2..............Field List......
T Total (ex VAT) £@@@.@@   (inc VAT) £@@@.@@   |TA / 1.15, |TA
```

then two values would get sent to the linking file. The REPORT utility asks how many columns there should be in the link file. Now enter the maximum number of register values printed on any subtotal or total line. In the example just given, the link file should clearly have two columns:

```
How many totals across? 2
```

After printing the report, the link file is created and the size of the file is displayed. The link file is produced even if the report is only displayed on screen, when 'S' is typed in at the 'Screen or Printer?' prompt.

There are a couple of problems associated with link files created using the REPORT utility. First, the file is arranged so that the first row of the link file is the total line, and the subtotal lines follow after that. Odd, but not necessarily a problem. For a file with three subtotals and a total, each line using five registers |A to |E, the rows and columns of the link file are arranged like this:

```
|TA     |TB     |TC     |TD     |TE
|SA1    |SB1    |SC1    |SD1    |SE1
|SA2    |SB2    |SC2    |SD2    |SE2
|SA3    |SB3    |SC3    |SD3    |SE3
```

Now what happens if the subtotal lines use only three registers?

```
|TA     |TB     |TC     |TD     |TE
|SA1    |SB1    |SC1    0       0
|SA2    |SB2    |SC2    0       0
|SA3    |SB3    |SC3
```

Most of the subtotal lines are filled out with zeroes, but ViewStore doesn't bother to fill out the very last subtotal line if it uses fewer registers than the total line. This means that ViewSheet can't read those values, '?Error' is displayed in spreadsheet cells that try. If this happens, then make sure that the subtotal lines print the same number

of registers as the total line. Do this by inserting extra @bbb patterns at the end of Half1, and matching them with |SZ in the field list. This won't actually make any difference to the report (if |Z remains zero, it will only print blanks), but it will fill out the link file.

The second problem with link files generated by REPORT arises when there are multiple subtotal lines:

```
T.Half1..........................
S
S Total    @@@@@@
S Average  @@@@@@
S
```

Each time a subtotal is needed, four lines are printed (two of them blank). But only one row of the link file is created. What happens is that the figure from the second subtotal line is put into the link file, but is immediately overwritten by the figure from the third line. So ViewSheet can only read the last figure. The only solution is to make sure all the important values are on the last subtotal line (not counting blank lines) in the report definition.

With both these ways of creating a link file, through LINK or through REPORT, the number of columns is fixed, either by the fieldnames typed in, or by the registers used in the report definition. However, the number of rows depends on how many records there are in the database, how many there are in the selection file, or how many subtotal and total lines are generated by the report.

Building a Spreadsheet System

A complete system for printing invoices has been chosen for detailed description, as it incorporates an extensive range of the advanced features of ViewSheet.

The system is intended to allow invoices to be typed in and printed out as simply as possible. Calculations for carriage costs, VAT and totalling are done automatically by ViewSheet.

In the example, the invoice total is made up of the sum of the costs for various items, plus a delivery charge, and value added tax. The item costs are the unit price for an item multiplied by the number bought (so

two units at £20 each makes an item cost of £40). Each type of item is allocated a carriage code indicating the cost of the method of delivery. The total delivery cost of an order depends on the item that is costliest to deliver. If an item costs £1 to deliver and another costs £2, the total delivery cost for the two items is £2.

The Invoice Spreadsheet

Figure 15.3 shows an almost completed invoice model. Its structure follows the description just mentioned. Up to 10 items purchased can be put in cells A7 to A16, their product codes in column B and the number bought in column C.

```
VM   SLOT=F18
         CONTENTS=CHOOSE(MAX(E7E16)+1,0,E28,E29,E30,E31)

0        ......A......B......C        D        E        F
      1 SMALL-T
      2
      3 INVOICE
      4
      5 Item      Code    Units    Price     Del.    Cost
      6
......7                             0.00     0.00    0.00
......8                             0.00     0.00    0.00
......9                             0.00     0.00    0.00
.....10                             0.00     0.00    0.00
.....11                             0.00     0.00    0.00
.....12                             0.00     0.00    0.00
.....13                             0.00     0.00    0.00
.....14                             0.00     0.00    0.00
.....15                             0.00     0.00    0.00
.....16                             0.00     0.00    0.00
.....17
     18 Carriag                                 >    0.00
     19 Sub-tot                                      0.00
     20 VAT                                           0.00
     21
     22 Total D                            £         0.00
```

Figure 15.3. The invoice model.

The formulae for the prices in column D use LOOKUP to find the product code in a table starting at A28, and they return the price from the table at B28. For example, the price formula in cell D7 is:

```
LOOKUP (B2, A28 A36, B28 B36)
```

A similar formula is also used for the carriage codes in cell E7, to return the corresponding code from the table starting at C28:

```
LOOKUP (B2, A28 A36, C28 C36)
```

233

This and the price formulae can be replicated downwards to the other nine cells available. The costs in column F are the totals for each line of the invoice, C7 * D7 for example, in cell F7.

The carriage cost in cell E18 is calculated by the formula shown in the status area of figure 15.3. This works by taking the maximum value of the carriage codes in the range D7 D16 (which vary between one and four for different items), and using this to choose one value from those held in a table starting at E28. Each carriage code from the price list table identifies a 'postage cost band', and the band charge is held in a separate table. The advantage of this approach is that the postage costs can be seen and changed more readily than if they are held within a formula or spread throughout the price list table. Figure 15.4 shows the price list and carriage band charge tables.

Note that there is a zero in the price list; this is so that an empty cell (value zero) chooses a zero price and zero carriage band. The zero carriage band is used in the CHOOSE function, to give a zero delivery cost. Without this value, the empty cells would cause '?Error' to be displayed in some formula cells.

The subtotal formula in cell E19 of the invoice model is of course the sum of the range E7 E18. The VAT is calculated as a proportion of this subtotal, using the percentage rate held in cell F28. Again, it is better to hold this rate as a value in a separate cell, than to embed it within a formula.

Finally, the invoice total is the sum of the subtotal and the VAT. Two further points to note are that the number format for window 0 is set to D2RM, and that all areas of the sheet except that for entering details of a sale are protected to prevent inadvertent alteration of the formulae.

```
0       .......A.......B.......C       D       E       F
    24
    25 PRODUCT    PRICE CARRIAG          DELIV'Y   VAT
    26    CODE    POINT  BAND             COST    RATE
    27
    28 100461    14.95      3             1.00    0.15
    29 100462     5.95      2             1.50
    30 100401     1.95      2             2.00
    31 100675     1.85      1             6.00
    32 100822     3.95      2
    33 100407     0.65      1
    34 100970     5.50      3
    35 100971     3.65      2
    36      0     0.00      0
```

Figure 15.4. Price list section of the invoice model.

Printing Out Simple Invoices

While the model calculates the amounts properly as it stands, it is
unacceptable as a record of any transaction. Were the model printed
out, labels such as SMALL-TIME GADGETS LTD. in cell A1 would be
truncated as they are on the sheet display. A set of printer window
definitions can be used both to present the necessary information in a
better layout, and to hide any unnecessary information.

The following represent a highly acceptable display:

Wi	TopL	BotR	Pos	Cw	Bw	Fmt	Opt
0	A1	I19		7	7	D2RM	
P0	A1	A4		50	7	FRM	TS12
P1	A5	A16	B0	24	7	FRM	TS
P2	C5	C16	R1	5	7	D0RM	TS
P3	D5	D16	R2	7	7	D2RM	TS
P4	F5	F16	R3	8	7	D2RM	TS
P5	A18	A22	B1	24	7	FRM	TS
P6	E18	F22	B3	8	7	D2RM	TS

Notice the variations in column width, and in the number format.
Obviously the money values are printed out with two decimal places,
but values like the number of each type of item sold should be printed as
whole numbers.

When these windows are set up, the window scheme can be tested by
returning to Command mode and using the SCREEN command. The
model is displayed on the screen, using a layout controlled by the printer
windows, and any small adjustments necessary can be made. It's a
good idea to put in some dummy data temporarily while this is done.
When the windows are correct, it is wise to save the window scheme
using the SW command. Finally, the whole model can be saved as a
mask: delete the example data, and a blank model consisting of just the
data, labels and formulae can be saved, using a name such as M.INVOICE
for example.

To prepare an invoice, load the M.INVOICE mask file, and enter the data
on each item purchased: a description, the number purchased, and each
item's stock code. Calculations and totalling are done automatically.
When the model is correct, return to Command mode. Load the
appropriate printer driver, then simply use the PRINT command to
produce an invoice. Figure 15.5 shows a typical result.

```
SMALL-TIME GADGETS LTD.

INVOICE NUMBER ......... 1

Item                    Units    Price     Cost

small bowl                2      3.65      7.30
lemon squeezer            1      1.95      1.95
chopping board            1      5.95      5.95
                                 0.00      0.00
                                 0.00      0.00
                                 0.00      0.00
                                 0.00      0.00
                                 0.00      0.00
                                 0.00      0.00
                                 0.00      0.00

Carriage                                   1.50
Sub-total                                 16.70
VAT                                        2.50

Total Due                         £       19.20
```

Figure 15.5. An example printed invoice.

Refining the Invoice

An invoice number is shown at the top of figure 15.5. This serial number increases by one every time a new invoice is created. To set this up is not a simple procedure, so follow the steps below exactly:

1) Open a link file to hold the serial number, using the command:

   CREATE 100 1 1

   This creates a file called 'V.VS100', and sets the serial number in the file to zero.

2) Load the mask file and enter Sheet mode. Unprotect row three using SHIFT–f6.

3) Enter the formula WRITE (100, 1, 1, –5) in cell B3, and then delete it with SHIFT–f9.

4) Now enter this formula in cell B3:

   WRITE(100, 1, 1, READ (100, 1, 1) + (B3 <= 0))

5) Restore the protection on row three. Now press RECALCULATE (key SHIFT–f7) until the value shown in cell B3 is zero. Once this is done,

make no further changes to the model. Return to Command mode and resave the mask file.

This procedure relies on a complicated circular reference. The way it works is as follows. The formula in cell B3 reads the serial number from the link file, then adds the conditional value (B3 <= 0) before writing the serial number back to the link file. But the value of cell B3 is the serial number written to the file, so the condition is false. Thus, nothing is added to the serial number, because false has the value zero!

But when the mask is first loaded to prepare an invoice, the cells contain the values they held when the mask file was saved. When saved, cell B3 contained zero. So the very first time the invoice mask is recalculated, cell B3 finds that B3 <= 0 is true, and thus adds one to the serial number. The important point is that this happens only once for each invoice.

Saving the mask with a zero value for cell B3 is accomplished by steps three to five above. The file is seeded with minus five value by the dummy formula in cell B3, and is incremented to zero by recalculating the final cell B3 formula.

Printing the serial number on the head of the invoice, as shown in figure 15.5 can most easily be done by altering the definition of printer window zero as follows:

```
Wi TopL  BotR  Pos Cw  Bw Fmt  Opt
P0 A1    B4        24  7  D0LM TS2
```

and editing cell A3 to read 'INVOICE NUMBER..........'.

For a really professional-looking result, more text might be required at the top of the sheet, giving the address of the company and so on. This could be typed onto the spreadsheet in the same way as the company name. But a better result can be had by using the VSXFER spooler from Chapter Six instead of printing the invoice directly. The spooled file can then be merged with a standard page from VIEW and printed. This can all be done via an exec file or even a function key definition, so all that's necessary is to load the invoice mask, add the product data, then *EXEC the file or press the function key.

237

The file or key definition needs to mimic the following commands:

```
PRINTER VSXFER
PRINT
INVMOD
*WORD
NEW
LOAD INVTOP
READ INVMOD
READ INVBOT
PRINTER FX80
PRINT
PRINT
*SHEET
```

This uses a spool file INVMOD to transfer the model from ViewSheet to VIEW. The other files INVTOP and INVBOT contain standard text for the top and bottom of the invoice page, of which two copies are printed. One file might include the company address details, the other an explanation of the trading terms and conditions.

Small Errors in ViewSheet Mathematics

One of the most perplexing problems with ViewSheet is the so-called 'small errors' problem. For a demonstration, type NEW to clear the spreadsheet, then go to Sheet mode and edit the default window definition to give a column width of 15. Now enter the following formula in cell A1:

```
6 + 0.3
```

The value of the formula should clearly be 6.3. Many fractions can be expressed exactly in base 10: a twentieth is 0.05, a fifth is 0.2 and an eighth is 0.125. But machines can't always use as many decimal places as are needed. Imagine trying to express one eighth with only two significant figures - 0.13 is the closest it is possible to get; rounding is necessary. It's a bit like that with ViewSheet. It can only use 32 significant figures in binary. The nearest it can get to the true figure of 6.3 is 6.300000001. Try 7.8 too.

The window or cell formats can be used to control this problem. If a number should be an integer, with no fractional part, use D0RM. This doesn't change the actual value in the cell, but it stops any tiny error in that value being displayed. Alternatively, use of the INT function at

strategic points in the model may be useful. Be careful with INT, as INT (A1) won't give the rounded value but a truncated one: INT (A1 + 0.5) gives the rounded value of A1.

Error Messages

There are really three sorts of errors that may appear on the sheet.

The most common is when a '%' symbol appears in a cell. This means that the value of the cell cannot be displayed using the current display format. Either widen the columns, or change the display format.

At other times '?Error' appears. The full error message appears in the status area when the cell cursor is moved to the slot in question. Common messages include 'Division by zero', 'Lookup' and 'Propagated'. These messages may occur for two reasons. Sometimes there really is an error in the construction of the sheet, and a formula may need to be modified. But often they occur when preparing masks; because there is no data, many cells have a zero value, and these zeros can cause the problems. When data is added, the errors simply disappear. This is a good reason for preparing masks with dummy data, and then only deleting the data once the model works.

Program Listing 15.1

```
10 REM RWLINK Read/write Link files
20 REM by Graham Bell
30 REM for BBC B/B+/M/C/E
40 REM (C) Graham Bell 1988
50 :
60 PRINT "RWLINK"
70 ON ERROR PROCerror(0)
80 :
90 REM open Link file
100 handle% = FNopen("100")
110 :
120 REM equivalent of WRITE(100,1,1,0)
130 PROCwrite(handle%, 1, 1, 0)
140 :
150 REM equivalent of READ(100,1,1)
160 PRINT ' FNread(handle%, 1, 1)
170 :
180 REM close Link file
190 CLOSE# ?handle%
200 END
```

```
210 :
220 :
230 :
240 DEF PROCwrite(f%, x%, y%, n)
250 IF (x% < 1) OR (x% > f%?1) THEN PROCerror(1)
260 IF (y% < 1) OR (y% > f%?2) THEN PROCerror(2)
270 PTR# ?f% = 6 * ((y% - 1) * f%?1 + x%)
280 PRINT# ?f%, n
290 ENDPROC
300 :
310 DEF FNread(f%, x%, y%)
320 LOCAL n
330 IF (x% < 1) OR (x% > f%?1) THEN PROCerror(1)
340 IF (y% < 1) OR (y% > f%?2) THEN PROCerror(2)
350 PTR# ?f% = 6 * ((y% - 1) * f%?1 + x%)
360 INPUT# ?f%, n
370 = n
380 :
390 DEF FNopen(file$)
400 LOCAL f%
410 DIM f% 2
420 PRINT ' "Opening V.VS"; file$
430 ?f% = OPENUP("V.VS" + file$)
440 IF ?f% = 0 THEN PROCerror(3)
450 PTR# ?f% = 5
460 IF BGET# ?f% <> &AA THEN PROCerror(4)
470 PTR# ?f% = 3
480 f%?2 = BGET# ?f%
490 f%?1 = BGET# ?f%
500 =f%
510 :
520 DEF PROCerror(number%)
530 CLOSE# 0
540 IF number% = 0 THEN REPORT
550 IF number% = 1 THEN PRINT "x coordinate too big"
560 IF number% = 2 THEN PRINT "y coordinate too big"
570 IF number% = 3 THEN PRINT "Can't find file"
580 IF number% = 4 THEN PRINT "Not Link file"
590 END
```

Listing 15.1 Reading and writing link files.

16 : ViewStore Hints and Tips

!BOOT files

An exec file can ease the process of starting up ViewStore. This is exactly the same as the process described in Chapter 13 for ViewSheet. A text file on a disc can contain the text that would normally be typed in to begin a work-session with the database. By using this file with the *EXEC command, or by auto-booting the file called !BOOT, the computer can act as if the text had been typed in at the keyboard.

The best way to construct an exec file like this is to begin by starting a ViewStore session in the normal way. Each time something is typed, write everything down. Include all keypresses - RETURN is particularly important. Now leave ViewStore, and use a wordprocessor or the *BUILD command as described in Chapter 13, to type in the text, exactly as it was noted down. When saved using the name !BOOT, type in '*FX 4,3'. This sets the boot option; it means that the !BOOT file will be used with *EXEC. Once set up in this way, the !BOOT file can be used simply by pressing SHIFT–BREAK. Conversely, *FX 4,0 means the !BOOT file will be ignored.

A typical ViewStore !BOOT file might look like figure 16.1. This file enters ViewStore, sets up the screen mode, the prefixes, and loads the database. The *FX 202 command has the same effect as pressing CAPS LOCK to turn caps lock off. Of course, by using the DECODE program from Chapter 13, the screen colour can be set too. One important consideration is that the !BOOT file always has to be on the disc in drive zero, so in this case, it should be on the same floppy disc as the database file itself. The ViewStore utility disc goes in drive one.

```
*TV 0,1
*FX 202,48
*STORE
MODE 3
PREFIX D :0.
PREFIX F :2.
PREFIX U :1.
PREFIX I :2.
PREFIX S :3.
LOAD BIBLIOG
```

Figure 16.1. Typical ViewStore !BOOT file.

Limited Access

Quite often, a database may be set up to be used by more than one person. It may be that different people need access to different parts of the database. Or maybe only certain parts of the database need frequent updating. Or perhaps the information is sensitive, and some users should have only limited access to the data. It is sensible to limit the accessibility of parts of the database for the following reasons: convenience, sensitivity, and security.

It is simpler to be presented with only the important and relevant information. If the addresses of several companies on a database need to be changed, then a screen showing details of their credit standing is confusing. It is possible to use several different card or spreadsheet layouts, each tailored to doing a different sort of job on the database. One card might be arranged for entering the initial details of a new record, one might be designed for doing the regular monthly amendments to credit standings, a third could be for amending address details. The cards could show different fields depending which are most important for each job.

The second reason for limited access to the database is sensitivity. This doesn't mean keeping sensitive information on the database away from prying eyes, nor the prevention of deliberate tampering with the data. ViewStore is not that sort of database - its files are not encrypted, it is not password protected, it is not meant for 'multi-user' operation on a network. Only physical security, putting the database discs somewhere safe, can maintain the confidentiality of the information. But it can at least be made difficult for the casual user to see restricted and sensitive parts of the database.

Net: Consult the network manager about the implications of
                keeping any sensitive information on a network. It can't
                be kept fully secure using Econet, and this may violate the
                requirements of the Data Protection Act.

There remains the problem of altering the details of a record. How easy
is it to alter one of the fields by pressing the wrong key accidentally,
when browsing through the database? Fields apparently starting with
the characters '\', '_' or ']' are often caused by this, because of those
keys' proximity to the cursor keys. Again, a card or spreadsheet layout
that allowed only the relevant information to be changed could ease
matters.

This can all be done by having more than one format file for a database.
The format file contains all the details from the record format, the
database header and the card layout. So by using a different format file,
the database could appear completely different on the Card display, in a
different screen mode, even with different field names or prompts.

Creating a New Format File

At the very beginning, the database has to be created by using the SETUP
utility. This puts a database file in the 'D' directory, a format file in the
'F' directory, and possibly one or more index files into the 'I' directory.
But it can only create a single format file.

When the database is set up properly, this master format file will contain
details of all the fields in each record, their fieldnames, their widths,
their places on the card display, even their numerical high and low
limits. A brand new format file is best created by using a copy of the
master format file. A copy could be made using the *COPY command, but
ViewStore has special commands to load and save format files.

To create a second format file, first load the database in the usual way.
Then use the command:

```
SF <filename>
```

to save the format in the file named. As usual, the F prefix is used in
naming the file, but this can be overridden by typing in the full name.

Assuming the F prefix was set to ':2.', then:

```
SF ADDRS
```

would save a new format file called ':2.F.ADDRS'. But a file called ':1.G.ADDRS' could be saved by typing:

```
SF :1.G.ADDRS
```

At this stage, the new file is exactly the same as the original, master format file. Any number of copies of the master format file can be made in this way.

It is important to keep the master format file unchanged. It is a good idea to 'lock' the file, using the *ACCESS command.

DFS	*ACCESS <filename> L
ADFS / Net	*ACCESS <filename> WRL

A locked format file can be used in the usual way, except that it can't be modified. So if the database header were changed, for example, then pressing ESCAPE would give a 'Locked' error message. Pressing any key after that returns to ViewStore Command mode, but the new details are not saved in the locked format file.

Using the Second Format File

When there are two format files, which one is used when loading the database? Taking the example of the Membership database from Chapter 11, the files were called:

D.MEMBERS	Database file
F.MEMBERS	Master format file
F.ADDRS	The new format file

Naturally, ViewStore tries to use the format file matching the name of the database file, unless another name is specified:

LOAD MEMBERS	Loads D.MEMBERS and F.MEMBERS
LOAD MEMBERS ADDRS	Loads D.MEMBERS and F.ADDRS

As an alternative to naming the file when the database is loaded, ViewStore provides an LF command to change the format file at any time:

```
LOAD   MEMBERS
LF  ADDRS
```

The original master format file can be loaded again later by entering the command 'LF MEMBERS'.

Payroll Database Example

Figure 16.2 shows part of the main record format for a company payroll. It is divided into three major sections, the employees' names and personal details, their addresses, and their salaries. Each of these sections is labelled on the Card mode display with its own dummy field; PERSONAL, ADDRESS and SALARY. The card display is shown in figure 16.3.

```
L Space 29278
Record format
Field name

Field name......Wid.T.S.D.Low limit..High limit.I.Key.Index name
First name      10 A Y
Surname         10 A Y                              Y    5 SURNAME
Job Title       15 T Y
Department       8 A Y
Born             8 D N
Joined           8 D N
Current £        5 N N 0 1000      50000
Commission       3 A N
Previous £       5 N N 0 1000      50000
Last rise        8 D N
Address 1       20 T Y
Address 2       20 T Y
Address 3       20 T Y
Address 4       15 T Y
Phone           12 T N
PERSONAL         0
SALARY           0
ADDRESS          0
```

Figure 16.2. Payroll database master record format.

```
              PERSONAL                        ADDRESS
        First name _____       Address 1 _____
           Surname _____       Address 2 _____
                                     Address 3 _____
              Born _____         Address 4 _____
            Joined _____             Phone _____

                                SALARY

         Job Title _____    Current £ _____    Previous £ _____
        Department _____       Commission ___    Last rise _____
```

Figure 16.3. Payroll database master card layout.

This layout is fine for the person in charge of the database, perhaps a personel manager. But simply keeping the address records up to date can be done by someone else without necessarily giving unrestricted access to salary information. The principle is to have different format files for different levels of access.

A second format file can be created using SF, loaded using LF, and modified to hide some of the information. The modified format is shown in figure 16.4. Fields can't be deleted from the record format using SHIFT–f9. To hide the information itself, all that need be done is to set the width to zero. This makes any field disappear from the spreadsheet display, and prevents the data from being modified. But as with dummy fields, the field name is still shown in Card mode. To make this disappear too, just delete it using DELETE END OF FIELD (key f3). This has been done to nine of the fields in the modified format, leaving only the name and address fields. Accordingly, the sample record card shown in figure 16.5 fails to show the complete record. Notice that even the dummy field, SALARY, has been hidden and another NAME, has been modified.

```
L Space 29342
Record format
Field name

Field name......Wid.T.S.D.Low limit..High limit.I.Key.Index name
First name       10 A Y
Surname          10 A Y                                  Y   5 SURNAME
                  0 T Y
                  0 A Y
                  0 D N
                  0 D N
                  0 N N 0 1000        50000
                  0 A N
                  0 N N 0 1000        50000
                  0 D N
Address 1        20 T Y
Address 2        20 T Y
Address 3        20 T Y
Address 4        15 T Y
Phone            12 T N
NAME              0
                  0
ADDRESS           0
```

Figure 16.4. Modified database format, showing hidden fields.

```
L Space 29342 Indexed by entry
                          EMPLOYEES
Home phone number:

                    NAME                         ADDRESS

First name Patricia              Address 1 7 Trinity Terrace
    Surname MacDonald            Address 2 Chorlton-cum-Hardy
                                 Address 3 Manchester
                                 Address 4 M21 5QZ

                                 Phone
```

Figure 16.5. Card mode display using the modified format.

With this payroll database, it may be best to name the edited format file
the same as the database file, so that it is used by default. The master
format file can be renamed, so that it requires a positive act to load it in
with LF.

Two important warnings. First, this payroll database is only a record of
employees' salaries. So simple a database could not be used to replace a
proper payroll accounting system. Second, limiting access in this way
does not prevent access to the hidden fields; it merely makes it slightly
more difficult for the casual user.

The idea of having various levels of access to a database can be extended. It's possible to have several format files, each hiding or revealing different parts of the same database, each tailored to the needs of a different task.

Joining Databases Together

Another way of using a second format file allows brand new data to be added to the database, without access to the database itself! This involves putting the new data into a miniature version of the database file, which can then be added to the main database later.

The general procedure involves using the SETUP utility to create a blank database, on a new disc. When set up, the format file can be deleted, and then replaced by a copy of the format file from the main database. The procedure is shown in figure 16.6, using a main database called MAIN and a new database called MINI. The command SF MINI overwrites the format file F.MINI that SETUP created, and replaces it with a copy of F.MAIN.

```
=>UTILITY SETUP
SETUP
Set up database or report (D,R)? D
Filename of database? MINI
Format filename (database name)?
How many bytes to reserve? 0
Number of indexes? 0
Creating Format file
Creating Database file
Database set up
---change discs or directory---
=>LOAD MAIN
---change discs or directory---
=>SF MINI
=>LOAD MINI
=>
```

Figure 16.6. Creating a miniature database.

DFS: As MINI and MAIN should be on separate discs, changing discs will be necessary.

ADFS/Net: Changing discs on ADFS involves the use of the *MOUNT command; changing directories may be necessary too, using *DIR.

248

Net: The miniature database could even be created using a different user name, to avoid contact with the main database. Consult the network manager.

Once the MINI database is loaded, switch off all indexes, by putting 'R' or 'N' in the record format I column. There is no point in having any index files, and switching them off saves a lot of time while entering new data. When the miniature file is set up properly, new records can be entered without any reference to the main part of the database. Simply load it in and use it just as if it were a complete database. It is even possible to have several miniature databases, all being worked on at once.

The only problem is how to join the main and miniature databases together. This can be accomplished with the JOIN program listed at the end of the chapter.

The JOIN Program

Superficially, all that is required to merge two databases is to append one file to another. There are numerous ways of doing this, and programming toolkits often provide a method of concatenating files: Computer Concepts' Disc Doctor implements a *JOIN command. However, this ignores the special ViewStore file structure. All ViewStore database files include an *end of data marker*, but the concatenated file would contain two!

JOIN is a BASIC utility program enabling one or more miniature database files to be appended to a main ViewStore database, retaining the correct file structure. Great care should be taken that the mini databases are created using a format compatible with the main database. There are no limits on the maximum size of fields, records or files, save those imposed by ViewStore and the filing system in use. Deleted records in the miniature update file are not added to the main database.

The program first prompts for the prefix to use for all the data files. This prefix works in exactly the same way as the ViewStore prefix, and it can be overridden by entering a full filename at any point.

The second question is as follows:

```
Name of main database?
```

249

Enter the filename of the main database to which the records will be added. Remember that the prefix may be added to the filename. The program opens the main database file, and carries out a short check to ensure that the file is indeed a ViewStore database. If it isn't, an error message is printed, and the program ends.

Now the program asks for the name of the first mini database:

```
Name of extra data file?
```

Enter the filename. The new file is opened, and tested to see if it is a valid database. This test can't ensure that the field types in the mini file are compatible with the main file. When the mini file is checked, confirm that the records in the file should be added to the main database. At this stage, if 'N' is pressed, then the file will be ignored. If "Y" has been pressed, the program reads the records from the mini file, and appends them to the main database. The number of records transferred in this way is shown on screen when the process is completed.

DFS: Great care is needed to prevent the 'Can't extend' error. It will usually be necessary to have the main and mini databases on separate discs, or on different sides of the same disc with a single disc drive. For example:

prefix	:0.
main database	MAIN
mini database	:1.D.MINI

The main file is called ':0.D.MAIN', and the mini file is ':1.D.MINI'. Using a complete name for the mini file overrides the prefix.

Finally, the program returns to the 'Name of extra data file?' prompt, and a further small file may be added to the main database. Any number of files may be joined in this way, but take care not to add two copies of the same file. And don't add a format file onto the end of a database! Mistakes are possible, and so as always, a back-up is vital. When all the small files have been added to the main database, just press RETURN at the 'Name of extra data file?' prompt. This closes all the files and ends the program.

When a file has been added to a main database, the index files for the database will be out of date. They need to be rebuilt. Load the newly

enlarged database and its master format file, then use the INDEX utility to recreate all the indexes as follows:

```
=>UTILITY INDEX
INDEX
Use select file (N,Y)? N
Field name? *
=>
```

The * wildcard specifies all the index files. ViewStore reconstructs the index files for the database, and includes the new records added with the JOIN COMMAND.

Entering JOIN

The JOIN program shares many procedures with the PURGE program from Chapter 11, PROCeod_pointer, FNname, FNconfirm, FNopen, and parts of PROCerror. These need not be typed in twice. First, type in the unique part of JOIN, up to line 500, and save the part-completed program. Next load the PURGE program, and delete the unique parts up to line 1960. The remaining part of PURGE can be added to JOIN. Type:

```
RENUMBER 5000
*SPOOL  SHARED
LIST
*SPOOL
LOAD "JOIN"
*EXEC  SHARED
RENUMBER
```

This creates a file called SHARED, which contains the shared procedures, and these are then added to the end of JOIN.

JOIN works by opening the main database file, and reading it backwards, searching for the ViewStore end of data marker in the file. When found, the mini file can be opened, and data copied from one file to the other until the end of the mini file. The original end of data marker gets overwritten, but the one from the mini file is copied to the end of the extended main database. The ESCAPE key is disabled whilst the database is being altered, but BREAK should not be pressed while the program is running. If it is, then the files may be left open and the database corrupted. The back-up copy of the database would have to be used.

251

ViewStore and VIEW

Anything ViewStore prints can be transferred to VIEW by loading the VSXFER printer driver described in Chapter Six. Reports, labels and the printed output of any of the utilities, all can be put into a file using VSXFER, then read into VIEW. This is most useful with the LABEL or REPORT utilities, where the output may need to be edited before final printing, or merged with other text to form a larger document. For reports in particular, the more sophisticated page layout features of VIEW may be needed too. The MACRO utility described in Chapter 11 allows direct transfer of data to VIEW.

But VIEW can also read whole ViewStore databases directly. In VIEW, a database file can be read in using the READ command. Figure 16.7 shows part of the Bibliography database read into VIEW. Notice how the tab stops of the ruler have been arranged to align the fields of the database in neat columns. This works because ViewStore uses TAB characters to mark the end of a field.

```
F I...............*...................*...*......*...*....................<
   JOSEPH,Richard   By the Book                AU  1.6.87  95 book View typesetting dtp
   WILLIAMSON,Clive Soft Options               AU  1.6.87 102 printer dtp fonts
   HARRISON,Mike    New Tune on an old Fiddle  PCW 1.5.87 160 wordprocessor flowchart
   MALCOLM,Peter    Caught in the Net          PCW 1.5.87 134 network multi-user
   LIARDET,Mike     Branching Out              PCW 1.5.87 166 Prolog AI language
   LIARDET,Mike     Balancing the Scales       PCW 1.4.87 152 Prolog AI lists language
   PELLI,Denis      Programming in PostScript  B   1.5.87 185 dtp postscript printer
   SEYBOLD,John     Desk Top Publishing        B   1.5.87 149 dtp
   ***********************************************************************************
```

Figure 16.7. ViewStore database read into VIEW.

Difficulties will sometimes arise in reading database files, when the amount of data in a single record exceeds about 125 characters. VIEW allows a maximum of 132 characters per line, and if more are read, then VIEW splits the line into two.

Tables written in VIEW and laid out like this, can also be converted back into ViewStore databases using IMPORT. Unlike the other utilities supplied with ViewStore, IMPORT is a BASIC program. Enter BASIC and run the program. It asks a series of questions about the file to be imported, then reads it in and creates a ViewStore database. Figure 16.8 shows the sequence of questions and the answers necessary to import a VIEW file like that in figure 16.7. A TAB must separate each of the fields within a line, and each line of the text file produces one database record. It is best to delete the ruler beforehand.

```
> CHAIN "IMPORT"
ViewStore file conversion V1.1
Source file                         : VIEWTXT
Destination file                    : D.NEWDB
Position in file where data starts  : 0
Record separator                    : 13
Appears before first record  (Y,N) ? N
Appears after last record    (Y,N) ? Y
Field separator                     : 9
Appears before first field   (Y,N) ? N
Appears after last field     (Y,N) ? N
End of file marker                  :
Is the data reversed         (N,Y) ? N
Leading character to skip           :
Trailing character to skip          :
ViewStore record size               : +20
Importing VIEWTXT
>
```

Figure 16.8. Importing a VIEW file.

When IMPORT finishes, it displays the number of records that have been read and transferred into the destination file. IMPORT doesn't create a format file for the newly imported database. The easiest way to create one is to save one using the SETUP utility, or use the SF command from another database. The new database can be loaded into ViewStore and the format file modified to match the new field structure.

The second sort of VIEW file that can be imported is a macro file-that is a file consisting of nothing but macro calls. The important point is that there must be nothing in the file except macro calls, not even a ruler. For example:

```
F I.......*.......*.......*.......*.......*.......*.......*.<
AA Brian Hammond,14 Temperance Terrace,Rickmansworth,Middx.
AA Jessica Ellis,<Flat 7,Suffolk Mansions>,Stokeley Green,London NW3
AA Henry James,2 The Crescent,Woking,Surrey
***************************************************************
```

The answers to the questions posed by the IMPORT utility to import a macro file are shown in figure 16.9. The field separator must be a comma. The record separator must include the name of the macro, in this case 'AA'. Sadly, IMPORT can't deal with the case where a macro parameter contains a comma, as in the second line above, although the parameter is enclosed in angle brackets, IMPORT treats this as two fields.

```
Position in file where data starts : 0
Record separator                   : 13,128,"A","A"
Appears before first record  (Y,N) ? Y
Appears after last record    (Y,N) ? N
Field separator                    : ","
Appears before first field   (Y,N) ? N
Appears after last field     (Y,N) ? N
End of file marker                 :
Is the data reversed         (N,Y) ? N
Leading character to skip          :
Trailing character to skip         :
ViewStore record size              : +20
```

Figure 16.9. Importing a VIEW macro file.

The ViewStore manual describes the procedure necessary to import files from Ashton-Tate's dBase II and Acornsoft's Database. In fact, the answers given for dBase II can be used to import almost any database, providing it can be turned into an ASCII file, one record per line, with commas between fields.

Adding New Fields

With an established database in regular use, the need for an extra field will inevitably arise. Adding the new field is an easy job, providing that it can go last in the record format. This means that it will appear at the very end of the Spreadsheet mode display, but of course the Card mode display can be altered to place the new field anywhere.

If the extra field is to go at the end of the record format, then adding the new field is simply a matter of loading the database, and editing the record format (key f1). Press CTRL–DOWN to go to the last line, and enter the details for the new field. Press f0 to go back to the data screen, and all that remains is to enter the data in each record for the new field.

The rarely noticed CURSOR LOCK function (key SHIFT–f2) may be vital here. Usually, there is an 'L' flag in the upper left-hand corner of the data screen; this indicates the field cursor is 'locked' to a particular field. Pressing DOWN moves the field cursor to the corresponding field of the next record. It stays in the same column of the Spreadsheet mode display. Now press SHIFT–f2, so the 'U' flag - 'unlocked'- is shown, and press DOWN again. The field cursor moves down to the next record, but instead of staying in the same column it jumps to the first field of the record. For adding data to one particular new field, obviously the

cursor should remain locked to that field. Remember that RETURN doesn't need to be pressed, just type in the new data and press DOWN.

Adding a new field somewhere in the middle of the existing fields, say putting a post code field between the last address field and the telephone number, is more tricky. The CONVERT utility is the only way to do this. If the fields are displayed using the LIST command:

```
1 Name          2 Address 1
3 Address 2     4 Address 3
5 Phone
```

the new field can be added between field 4 and field 5 with the procedure shown in figure 16.10. Notice that a '?' wildcard can be used to specify all the address fields together; they are transferred to the new file in the same order as in the old file. Once the database is fully converted, LIST would produce:

```
1 Name          2 Address 1
3 Address 2     4 Address 3
5 Postcode      6 Phone
```

Well, actually, it won't! But it would if the record format were altered to match the structure of the new data. CONVERT does not alter the format file to match the new field order. It also doesn't keep the indexes up to date, so all the indexes have to be rebuilt with the INDEX utility. They can all be done at once, as already shown.

```
=>UTILITY CONVERT
CONVERT
Use select file (N,Y)? N
Field 1? Name
Field 2? Address ?
Field 3? Postcode
Field 4? Phone
Field 5?
New record size (+20)? +30
New file name? NEWADDR
Converting records........
24 records processed
Largest record was 166 bytes
=>
```

Figure 16.10. Adding an extra field to a database with CONVERT.

Extra fields can't just be slotted into the record format in the middle. They can only be added at the end. This presents a major problem when CONVERT is used to alter the field order, or when new fields are added.

But don't format files have a structure similar to a database files? Can't database files be read into VIEW? And can't they be edited, then imported back to ViewStore? Figure 16.11 shows the Bibliography database's format file read into VIEW. Notice the new ruler at the top of the text. The tab stops are arranged to emphasise the similarity with the columnar layout of the record format. As with reading database files, VIEW places a 132 characters per line limit, but with format files this will only rarely be exceeded.

```
F I.......*.......*.......*.......*.......*.......*.......*.......*.......*.......*<
.. ............*...*....*...*..*..*..*..*.......*.......*....*.*.*.......*..............<
    Bibliography   S   +20   30   3
    AUTHOR         20   A    10   4   Y                          Y 8 AUTHOR Author of article:
    TITLE          30   T    11   5   Y                          R 8 TITLE  Title of article:
    MAG            3    A    54   4   N                          N          Magazine:
    DATE           8    D    53   5   N        1.1.70  31.12.99  Y 3 DATE   Date of publication:
    PAGE           3    N    53   6   N   0  1          300      N          Page number:
    KEYWORDS       30   T    8    6   Y                          N          Key words:
    **************************************************************************************
```

Figure 16.11. Bibliography format file read into VIEW.

When the format is in VIEW, it can be edited. The first line of figure 16.11 represents the database header; this is slightly different to the rest of the lines, which represent the rest of the record format. During editing, it's best to leave the actual lines alone, but their order can be changed to match the way the database file was amended with CONVERT. The MOVE BLOCK function is best for this. Extra lines can be inserted, and new format information typed in. Remember to separate the columns with TAB, not spaces. When everything is correct, delete the ruler line and save the VIEW version of the format file in a file called, say, NEWFORM.

The VIEW file can be imported back into ViewStore using the technique illustrated in figure 16.8. Enter BASIC and chain the IMPORT utility. The source file is the format file as modified by VIEW, NEWFORM. The destination is a new format file in the ViewStore 'F' directory, say F.NEWFORM. Don't overwrite the old format file F.BIBLIOG yet, in case the new format doesn't work as expected. Lastly, set the record size as +0.

Swap to ViewStore, and load the database using the newly imported format file:

```
LOAD BIBLIOG NEWFORM
```

Check out the new format by looking at the record format, the database header and the card layout. If all are as they should be then the old format file, F.BIBLIOG in this case, can be deleted, and F.NEWFORM renamed to take its place.

Fixing the Format

A trick that can be played with the format file is to add two extra bits to the database header. While the format file is in VIEW, add 'Y' and 'N' to the end of the first line. Compare figure 16.12 with 16.11 to see the difference. Remember the new columns must be separated from each other by TAB characters.

```
F I....*.......*......*......*.....*......*......*.......*.....<
.. ............*...*....*...*..*..*..*.*..*......*........*..*..*.*.......*...............<
  Bibliography  S   +20  30      3 Y N
  AUTHOR            20 A  10  4 Y              Y  8  AUTHOR  Author of article:
  TITLE             30 T  11  5 Y              R  8  TITLE   Title of article:
```

Figure 16.12. Fixing the Bibliography format file in VIEW.

When imported back into ViewStore and used as a format file, pressing f1 does not allow the record format to be inspected; the 'Fixed format' error message is displayed. This works because there are two 'hidden' fields in the database header. The second of these is called 'Allow format edit', and if it is set to 'N', then the record format can't be edited.

The first extra hidden field is called 'Allow multiple records'. If that is set to 'N' then the Card mode display only ever shows one card. Normally, ViewStore displays as many cards as can be fitted on a single screen. If the card is small, with few fields, then several can be shown at once. But if the first hidden field is set to 'N', then only one card at a time is displayed.

Automation

Repeating the same job frequently gets very tedious. If the database is used to print a big batch of labels every week, then the LABEL utility can be 'automated' with an *exec file*, sometimes also called a *command file*, since it contains only commands.

The best way to create a new exec file is to start by writing down everything typed in as a batch of labels are printed. It's a good idea to keep a note of the prompts too. Usually, the notes will look something like this:

```
PRINTER   JUKI4    (load printer driver)
UTILITY   LABEL
      Y            (use select file)
      P            (use printer)
      8            (lines per label)
      1            (between labels)
      35           (characters per label)
      0            (between labels)
      1            (labels across)
      Name         (first field on label)
      Address  1
      Address  2
      Address  3
      Postcode
      RETURN
      Y            (do alignment print)
```

Now use *BUILD or a wordprocessor to carefully type in just the responses. The completed file is shown in figure 16.13, as it appears in VIEW. Although label printing usually follows creating a selection file, it may not be worth automating the selection because the criteria used might be different each week. Of course, if they are always the same, then include them in the file too.

```
F  I.......*.......*.......*.......*.<
   PRINTER JUKI4
   UTILITY LABEL
   Y
   P
   8
   1
   35
   0
   1
   Name
   Address 1
   Address 2
   Address 3
   Postcode

   Y
   ****************************************
```

Figure 16.13. LABEL exec file in VIEW.

When the response text is typed in and saved in a file called, say, !LABELS, a new set of labels can be created at any time just by entering:

*EXEC !LABELS

It is a common convention to start all exec filenames with an exclamation, but not vital.

Master: Providing that !LABELS was created with *BUILD, or by using VIEW's WRITE command instead of SAVE, then '*EXEC !LABELS' can be shortened to '*!LABELS'. This works with the Watford DFS on a standard BBC B as well.

Most of the ViewStore utilities can be automated in a similar way, and a suite of exec files can make the day-to-day use of the database quick and very convenient.

Anything that can be put in an exec file can also be programmed into a function key. To give key f0 exactly the same effect as the exec file in Figure 16.13, the definition is:

```
*KEY 0 PRINTER JUKI4|MUTILITY LABEL|MY|MP|M8|M1|M35|M0|M1|M
Name|M Address 1|M Address 2|M Address 3|M Postcode|M|MY|M
```

Pressing f0 would then produce a whole batch of labels. Notice that the RETURN key becomes '|M' in the key definition which should all be typed on one line, pressing RETURN only at the end.

Clearly, a complicated key definition like this should be worked out beforehand, and the definition put in the '!BOOT' file.

259

Program Listing 16.1

```
10 REM JOIN Viewstore database
20 REM file joiner
30 REM for B/B+/M/C/2P/E + Viewstore
40 REM (C) Graham Bell 1987
50 :
60 REM ViewStore file constants
70 space% = &03
80 end_of_field% = &09
90 end_of_record% = &0D
100 end_of_data% = &01
110 deleted% = &02
120 pad% = &00
130 :
140 ON ERROR PROCerror (0)
150 PRINT "JOIN"
160 :
170 prefix$ = FNname ("Prefix for data files")
180 name$ = FNname ("Name of main database")
190 IF INSTR(name$, ".") = 0 THEN name$ = prefix$ + "D." +
    name$
200 IF NOT FNconfirm ("Have you made a backup") THEN GOTO 490
210 :
220 big_file% = FNopen (name$, "U")
230 big_eod% = FNeod_pointer (big_file%)
240 :
250 REPEAT
260 tiny_file$ = FNname ("Name of extra data file")
270 IF tiny_file$ = "" THEN GOTO 470
280 IF INSTR(tiny_file$,".") = 0 THENtiny_file$ = prefix$ +
    "D." + tiny_file$
290 tiny_file% = FNopen (tiny_file$, "R")
300 tiny_eod% = FNeod_pointer (tiny_file%)
310 IF NOT FNconfirm ("Add records") THEN GOTO 460
320 :
330 *FX 229,1
340 PTR# big_file% = FNeod_pointer (big_file%)
350 PTR# tiny_file% = 0
360 records% = 0
370 REPEAT
380 byte% = FNnext_byte (tiny_file%)
390 BPUT# big_file%, byte%
400 IF byte% = end_of_record% THEN records% = records% + 1 :
    PRINT ".";
410 UNTIL EOF# tiny_file%
420 IF POS <> 0 THEN PRINT
430 PRINT ; records%; " record(s) processed"
440 *FX 229,0
450 :
460 CLOSE# tiny_file%
```

```
470 UNTIL tiny_file$ = ""
480 CLOSE# big_file%
490 END
500 :
510 :
520 :
530 DEF FNeod_pointer (file%)
540 LOCAL ptr%, byte%
550 ptr% = EXT# file%
560 REPEAT
570 ptr% = ptr% - 1
580 PTR# file% = ptr%
590 byte% = BGET# file%
600 UNTIL (byte% <> pad%) OR (ptr% = 0)
610 REM only eod marker may preceed padding in valid file
620 IF (byte% <> end_of_data%) OR (ptr% = 0) THEN PROCerror(2)
630 = ptr%
640 :
650 DEF FNname (prompt$)
660 LOCAL name$
670 PRINT ' prompt$;
680 INPUT "? " name$
690 REM delete trailing spaces
700 REPEAT
710 IF RIGHT$(name$, 1) = " " THEN name$ = LEFT$(name$, LEN
    name$ - 1)
720 UNTIL RIGHT$(name$, 1) <> " "
730 = name$
740 :
750 DEF FNconfirm (prompt$)
760 LOCAL key$
770 PRINT prompt$;
780 INPUT " (N,Y)? " key$
790 key$ = LEFT$(key$, 1)
800 = INSTR("Yy", key$) <> 0
810 :
820 DEF FNopen (name$, control$)
830 REM requires Basic 2
840 LOCAL handle%
850 control$ = LEFT$(control$, 1)
860 PRINT "Opening "; name$
870 IF control$ = "R" THEN handle% = OPENIN name$
880 IF control$ = "U" THEN handle% = OPENUP name$
890 IF control$ = "W" THEN handle% = OPENOUT name$
900 IF handle% = 0 THEN PROCerror (1)
910 = handle%
920 :
930 DEF FNnext_byte (file%)
940 LOCAL byte%
950 byte% = BGET# file%
960 IF byte% <> deleted% THEN = byte%
970 REM read through deleted data
980 REPEAT
```

```
 990 byte% = BGET# file%
1000 UNTIL byte% = end_of_record%
1010 REM now get next valid byte
1020 = FNnext_byte (file%)
1030 :
1040 DEF PROCerror (number%)
1050 *FX 229,1
1060 CLOSE# 0
1070 IF number% = 0 THEN REPORT : PRINT
1080 IF number% = 1 THEN PRINT "Can't find file"
1090 IF number% = 2 THEN PRINT "File not a database"
1100 *FX 229,0
1110 END
```

Listing 16.1. JOIN database file joiner.

17: Advanced Reporting
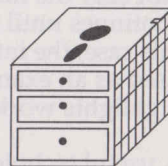
Highlights in ViewStore Reports

In ViewStore, highlights can used in report definitions to make the printer use different effects, such as bold and underlining. If highlight characters are put in the field list for a particular report line, then they will take effect when the report is printed out, providing the appropriate printer driver is used. The highlights are represented by '^1' and '^2'. For example, if a report is defined as shown in figure 17.1, then the printed report will have the Item name printed in bold. As the report line is printed, highlight 2 preceeding the Item field switches bold on, and the second highlight 2 in the field list switches it off again afterwards.

```
T.Half1.........................Half2.......Field list..........................
H Item         in stock     Price
R @@@@@@@@@@       @bbb      £@@.@@                   ^2,Item,^2,Stock,Price,A:Price*Stock
S
S    Sub-total value  £@@@@.@@                        |SA
S
T       Total value  £@@@@.@@                         |TA
```

Figure 17.1. Report definition with highlights.

Remember that items in the field list must be separated by commas; highlights are no exception. And remember to load the printer driver, with ViewStore's PRINTER command, before running the REPORT utility.

However, there are a couple of problems with ViewStore highlights. Taking this revised field list as an example:

```
Item,^1,Stock,^1,Price,A:Price
```

The first highlight is printed immediately before the stock field as expected, but the second highlight is *not* printed directly afterwards. Instead it is used immediately before the next field, the price information. This means the printed report may end up looking like the following:

```
Tee pieces      125    £ 1.84
```

Not only the number in stock is underlined; the underline effect continues until it is switched off at the beginning of the price field. In this case, the intervening pound sign is underlined too. Underline is used as an example of the problem because it is easier to see, the other highlights work in exactly the same way.

A useful technique to get around this problem is to use a *dummy comment*. This can be placed in carefully selected positions on the report line, to force the highlight sequences to be put in the correct place. The report line from figure 17.1 could be revised as follows:

```
T.Half1..........................
R @@@@@@@@@@@    @bbb @   £@@.@@
```

The matching field list is:

```
Item,^1,Stock,^1,^C1,Price,A:Price*Stock
```

The extra @ in Half1 simply causes a blank space to be printed in the report, because the comment is a dummy. But the important side-effect is that the space will be preceeded by the highlight code, so underlining will be switched off before the pound sign is reached:

```
Tee pieces      125    £ 1.84
```

The dummy comment ^C1 must be defined on an extra line at the end of the report. Clearly, if there are already other comments, then the next number in sequence should be used as the dummy:

```
T.Half1..............................
C Just press RETURN
```

When the prompt is displayed, just press RETURN as instructed, to ensure the comment gets printed as a blank.

This idea is also useful when you need to cancel one effect and start another immediately. A space is necessary between the two groups of highlights, and using @ together with a dummy comment provides this. It also means that ordinary text can be highlighted in a report. This:

```
T.Half1...........................
S   @Sub-total value @  £@@@@.@@
```

together with the field list '^1,^C1,^1,^C1,|SA' should print as:

```
 Sub-total value    £1275.38
```

Notice how the underlining can be made even on either side of the text, by offsetting the second @ by a single space. This matches the underlined space at the start of the underlining, caused by the first @.

A dummy comment can also be used to incorporate the '@' character itself into a report. It can't just be put in Half1 or Half2 because there it gets interpreted as a pattern. But if a pattern is used and a dummy ^C2 put in the field list, the comment could be used to print '@'. The comment line could be like this:

```
T.Half1............................
C Just press @ RETURN
```

There is a second problem associated with highlights. It is important that all effects are cancelled before you begin the next line. If a cancelling highlight lies at the end of the field list, this will *not* happen automatically. Another dummy comment could be used, but the best thing to do is to reserve one register to always be zero - |Z is probably the easiest to remember. Then assign zero to register |Z after any trailing highlights, to ensure they are printed. So the whole of the subtotal line could be printed underlined like this:

```
T.Half1............................
S    @Sub-total value      £@@@@.@@
```

using the field list '^1,^C1,|SA,^1,Z:0'. Adding zero to register |Z makes no real difference, but it does ensure the second highlight to end the underlining is used before the end of the line. If it just sits at the end of the field list ('^1,^C1,|SA,^1'), then it is ignored, and depending on your printer the underlining may carry over on to the next line.

When the report format is set up, print the report as usual using the REPORT utility. If a printer driver is used, the highlights will take effect, and parts of the report will be printed with the desired emphasis. If the VSXFER spooler from Chapter Six is used instead of a printer driver, then the highlights will go into the spooled file. VIEW can be used later, to read in the report, including its highlights, and incorporate it into a larger document. If the ASCII spooler from Chapter Six is used, then the highlights will be ignored.

Extended Highlights

The Acorn Printer Driver Generator includes an extra facility known as extended highlights. In VIEW, highlights 1 and 2 (keys SHIFT–f4 and SHIFT–f5) work as normal to give underlined and bold effects when used with a printer driver. But if highlight 2 is redefined using the HT edit command, like this:

```
HT 2 130
```

then a whole new set of highlights can be used.

Extended highlights can't be used at all with ViewSheet. Only the normal highlights 1 and 2 can be used, by including them in the OPT part of a printer window definition. This is described in Chapter 13.

Although figure 17.1 uses just highlights ^1 and ^2, there are in fact nine highlights in ViewStore, ^1 to ^9. The nine are not redefinable - there is no equivalent of VIEW's HT command. In the report definition field list ^1 and ^2 are used in place of the default VIEW highlights, to give the usual underline and bold effects. But in addition, using ^3 will mimic the effect of highlight 2 after resetting it to 130 with the HT command. The possible extended highlight sequences for VIEW and ViewStore are shown in figure 17.2. These extended highlight sequences can only be used with printer drivers created using the Printer Driver Generator.

```
VIEW        ViewStore
-           ^1           begin/end underline
***         ^3,^3,^3     begin/end bold
*-*         ^3,^1,^3     begin/end italic
*-          ^3,^1        begin subscript
**          ^3,^3        begin superscript
**-         ^3,^3,^1     end sub- or superscript
*--         ^3,^1,^1     begin/end alternate font
*---                     reset printer driver
--                       used with ViewIndex
```

Figure 17.2. Extended highlight sequences.

The extended highlights can be used in exactly the same way as simple highlights, so the field list:

```
^3,^1,^3,Item,^3,^1,^3,^C1,Stock,^1,^C1,Price,^1,Z:0
```

could be used to print the item name in italics, the number in stock
normally, plus the pound sign and price field underlined. Again blank
patterns, dummy comments and the register | Z are used to ensure the
highlights are printed in the right places.

Reports containing extended highlights can be transferred to VIEW in
the same way as normal reports, by printing them while using the
VSXFER printer driver. Note that the ^3 in ViewStore becomes highlight
2 in the VIEW file, and the 'HT@2@130' command must be inserted at the
beginning of the report once it is read into VIEW. In ViewStore it is
possible to use ^2 to control bold, even while using the other extended
highlights, but it isn't a good idea. Use ^3,^3,^3 instead, to maintain
compatibility with VIEW. The rule is to use ^1 and ^2 for underlining
and bold, or use extended highlights in figure 17.2, but don't mix the two
different methods.

Printing Without a Printer Driver

The routines described in Chapter 13 using special exec files to control
the printer, can all be used with ViewStore as well as with ViewSheet.
Most useful are the NLQ, POUND and CONDENS programs.

Editing Report Definitions

The biggest problem with trying to make changes to report definitions is
that new lines can't be inserted between existing lines. For example, if
the report definition looks like this:

```
T.Half1...............................
H Item          No. in stock     Page @@
H
R @@@@@@@@@@@@@@    @@@         £@bbb.bb
S Sub-total                     £@bbbb.bb
T Grand total                   £@bbbbb.bb
```

then the layout of the report can be improved by adding a few blank
lines around the subtotals. But nothing can be added between the R and
the S lines. ViewStore allows a line to be deleted, but not inserted.

It is a good idea to incorporate a few spare blank lines whenever you
devise a new report. However, blank lines can't really be included in a
report. The way to get around this is to insert several extra comment
lines into the report. Then before pressing ESCAPE, use f3 (DELETE END

OF FIELD) to delete the C in the type column. The report definition should
then look like figure 17.3. If this definition is used to print a report then
the REPORT utility just ignores the dummy lines, because they don't have
a type. Later, if lines have to be added to the report, then the dummy
lines can be replaced by real ones.

```
T.Half1................................
H Item            No. in stock       Page @@
H
  dummy line
R @@@@@@@@@@@@@@@    @@@          £@bbb.bb
  dummy line
S Sub-total                       £@bbbb.bb
  dummy line
T Grand total                     £@bbbbb.bb
```

Figure 17.3. Dummy lines in a report definition.

For more heavy duty editing, report definition files can usually be read
into VIEW for editing, just like whole databases or format files. Switch
to VIEW and use the READ command to read a report file called
'R.OLDREP' as follows:

```
NEW
READ R.OLDREP
```

Occasionally, very complex report lines with long field lists can't be
read successfully, because the line length exceeds the 132 character limit
in VIEW.

Figure 17.4 shows the report definition from the previous page read
into VIEW. A ruler can be inserted to tidy up the look of the file. It should
have three tabs, for the Half1, Half2 and Field list parts of the
definition. Remember the columns are separated by TAB characters.

```
I.......*.......*.......*.......*.......*.......*.......*.....<
.. ..*.......................................*....*..........................
H Item         No. in stock    Page @@        |P
H
R @@@@@@@@@@@@@@    @@@       £@bbb.bb      Item,'Num*',Value,A:'Num*'*Value
S
S Sub-total                  £@bbbb.bb       |SA
T Grand total                £@bbbbb.bb      |TA
***********************************************************************
```

Figure 17.4. New line added using VIEW.

When in VIEW, new lines can be added at will, using VIEW's INSERT LINE function (key f6). The important part to add is in the Type column; Half1, Half2 and the Field list can be added in VIEW, or they can be left until later and added when the report is transferred back to ViewStore. A single new subtotal line has been added in figure 17.4. Don't forget to get rid of the ruler line before saving the amended version. When the edited report definition is saved in a different file, say 'VIEWREP', the ViewStore IMPORT utility can be used to get the definition back into ViewStore. The series of answers for this is shown in figure 17.5. It's rather like importing a normal database or format file, but the ViewStore record size for a report file is 300 bytes. In the figure, the definition is imported into a file called 'R.NEWREP'.

```
=>*BASIC
>CHAIN "IMPORT"
ViewStore file conversion V1.1
Source file                           : VIEWREP
Destination file                      : R.NEWREP
Position in file where data starts : 0
Record separator                      : 13
Appears before first record   (Y,N) ? N
Appears after last record     (Y,N) ? Y
Field separator                       : 9
Appears before first field    (Y,N) ? N
Appears after last field      (Y,N) ? N
End of file marker                    :
Is the data reversed          (N,Y) ? N
Leading character to skip             :
Trailing character to skip            :
ViewStore record size                 : 300
Importing VIEWREP
>
```

Figure 17.5. Importing a report definition from VIEW.

Report definition files take up a great deal of memory because they have a record size of 300 bytes. Even a short comment line with just the prompt 'Today's date?' takes up 300 bytes. A moderately complex report with 20 lines would take up about 6k of memory! Although it is unlikely with a BBC model B+ or Master series micro, you may get a 'Record too big' or 'Memory full' error when using the REPORT utility. It is usually possible to switch to a different screen mode, perhaps mode 6 or 7. But you can squash the report definition file to leave more memory free, either while using IMPORT, or separately.

An imported report definition from VIEW can be compacted by answering one of the IMPORT program prompts differently, for example:

```
ViewStore record size: +30
```

makes the record size much smaller, yet still allows minor changes to be made to the definition when it's back in ViewStore. In this case, remember to add Half1, Half2 and the Field list for any new lines when you're still in VIEW - there won't be room to add them later.

Alternatively, the CONVERT utility can be used to compact a report definition file. Load the report definition into ViewStore:

```
=>LOAD  R.OLDREP  REPORT
=>UTILITY  CONVERT
CONVERT
Use select file (N,Y)? N
Field 1? *
Field 2?
New record size (+20)? +30
New file name? R.NEWREP
Converting records......
6 records processed
Largest record was 129 bytes
=>
```

This can reduce report definition files to less than a fifth of their original size, and yet still leave a little space for minor editing of the new definition, called in this case 'R.NEWREP'.

Mailshot Reports

A report definition can be quite complex. It can even include all the elements of a standard letter, so it is possible to produce a mailshot without involving VIEW at all.

Using the MEMBERS database described in Chapter 11, a report could be designed to print a letter inviting everyone to a regional meeting. The report definition might look like figure 17.6. Most of the details of this report should be familiar. There are a couple of H lines to leave a space at the top of each page. The page length is set to 66 lines by a P line, and two comments are used for the date and the cost of lunch. Take care over the comments - although the cost is a decimal number, a pattern like @@.@@ can't be used in Half1 to control the number format

because comments are always treated as text. The third comment is a dummy, used to control the placement of highlights.

```
T.Half1.........................................Half2..Field list.....
H
H
R @@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@           Street 1
R @@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@           Street 2
R @@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@           Street 3
R @@@@@@@@@@@@@@@@@@, @@@@@@@@@@@@@@@@@@           Town, County
R
R @@@@@@@@@@@@@@@@@@                               ^C1
R
R Dear @@@ @@@@@@@@@@@@@@@@@@@@@@@@@@@             Title, Surname
R
R The@Real Tennis Federation@is holding a regional   ^2,^C3,^2,^C3
R meeting in Hexham on December 9th, at the Assembly
R Rooms in Dere St.  If you wish to attend, could
R you return the form to the Federation Secretary
R as soon as possible.  A charge of £@@@@ will be     ^C2
R made for lunch and refreshments.
R
R Ron Shepherd, Secretary
R
R ---------------------------------------------------
R I wish to go to the regional meeting in Hexham on
R December 9th.  I will/won't require lunch at the
R Assembly Rooms.
R
R @@@@@@@@@@@@ @@@@@@@@@@@@@@@, Member number @@@@   'Fir*', 'Sur*',Number
R ---------------------------------------------------
S
C Today's date?                                      ^P
C Charge for lunch? £
C Just press RETURN
P 66
```

Figure 17.6. Invitation report.

When the report is printed, each of the R lines is printed for each of the records, so every person on the database will get the whole letter. And each letter will have their personal details inserted into it. Of course, an invitation like this would be combined with using the SELECT utility to select all those people who live in the north-east region.

Each letter is printed on a fresh page because a page eject (^P) is set in the S line. Page ejects can only be put in subtotal or total lines. The next thing is to ensure a subtotal gets printed at the end of every letter. This can be done by answering the prompts:

```
Sub-total field? *
Sub-total field?
```

This means that every field is used as a subtotal 'trigger', and so a sub-total is produced for every record.

271

VIEW and Multi-Macros

The letter in figure 17.6 could just as easily be produced using VIEW's macro facility. Producing a VIEW macro file from the records in a name and address database using the MACRO utility, in order to send out a mailshot, was covered in Chapter 11. But there can be a problem - VIEW is incapable of accepting more than 132 characters on a line. If VIEW tries to load a file containing a line longer than this, then the text can be corrupted.

In turn, this affects the ViewStore MACRO utility. As this extracts fields of information from a database record, it has to ensure that the fields don't add up to more than 132 characters. Occasionally, they will. If this happens, the MACRO utility splits the line, and the excess characters are placed on one or more additional lines below the macro. At the end, it displays how many macro calls have had to be split in this way.

The ViewStore manual suggests a remedy - the list of parameters for the macro should be edited in VIEW, joining the lines up into one again. But with a large database containing addresses of legal or financial partnerships, accountants or advertising agencies, this becomes impractical - they all seem to have long names!

One solution to this problem is to split a standard letter into a series of two or more macros. For an example, see the macro definitions in figure 17.7. When used together the two macros AA and AB would print out a complete letter, but each macro is incomplete on its own. This is a 'multi-macro'. Unlike a standard letter produced by passing parameters to a single macro definition, this multi-macro example would need two macro calls to complete it. The two must be in the correct order. Of course, two macros isn't the limit, you use several in a single multi-macro. A set of calls to print two of the letters is also given, in figure 17.8.

```
DM AA
CE Ball & Co. Heating Services
RJ @0
LJ @1
LJ @2
LJ @3
EM
DM AB
   @0

CE New Improved Specification

   We wish to bring to your attention our new
   'Executive' range of sealed system domestic
   central heating boilers...

   Yours @1,
   Edwin Ball
PE
EM
```

Figure 17.7. Multi-macro definition in VIEW.

```
AA 21.6.87,Moseley Construction,Cucumber Road,<Keighley, W. Yorks>
AB Attn: Installations Manager,faithfully
AA 21.6.87,17 Whin Terrace,Thornthwaite,Co. Durham
AA Attn: Joseph Addyman,sincerely
```

Figure 17.8. Two calls to the multi-macro in figure 17.7.

All the parameters of a multi-macro can add up to more than 132 characters, providing that any one individual macro doesn't have more than this number in its parameter list. So two macros with 100 characters-worth of parameters each is fine, but one macro with 200 is not. The other limit on macros is that there may be only 10 parameters - @0 to @9. This is less often a problem, but it can also be solved this way.

ViewStore and Multi-Macros

The MACRO utility can only produce a series of calls to a single macro, one call for each record in the database or one for each record in the SELECT file. The program listed at the end of this chapter can produce a macro file containing a series of calls to a multi-macro, which can be composed of several individually named macros.

Type in the BASIC program at the end of the chapter. Save it and then run it. The BASIC program should assemble and save a machine code file called 'VSMACRO'.

There are two steps in producing a file full of multi-macro calls from a database using VSMACRO. First a ViewStore report definition is set up, describing the fields to be taken from the database and put into the

macro parameter lists. The report definition has a special layout that is described next. Then, in ViewStore Command mode, VSMACRO and the database are loaded, and the REPORT utility is used to 'print' a report, using the special report definition file. VSMACRO creates the macro file from the report.

Multi-Macro Report Definitions

The first operation is to create a blank report definition file, using the SETUP utility as normal. Load the new blank definition file, and switch to the data screen to see the blank definition.

One report line in the definition is needed for each macro - that means two or more lines for a multi-macro. Obviously, the macros in the report definition should be in the order they are required in the multi-macro. There are three types of information to be entered for each macro - Type, Format line and Field list.

Type must be R for every macro. There must be no other types of line except comments - no header lines, sub-totals or totals.

The Format line, placed in the fields called Half1 and Half2 as normal, must start with a two-letter name for the macro, as used in the macro definitions. This must be different for each macro. Immediately after the macro name, the @@@@ patterns should be listed. Each pattern will be replaced by information from the database when the macro file is written. Separate the patterns with commas, not spaces. If the pattern may be replaced by information that might include a comma, then surround the format with angle brackets. These are necessary for VIEW to interpret the macro parameters correctly. Finally, put an extra four character pattern at the end.

A complete Format line could look like this:

```
    T.Half1.........................................
    R AC@@@@@@@@@@@@@@,@@@.@,<@@@@@@@@@@@@@@@@>,@@@@
```

The macro AC has three parameters - the second is a number with one decimal place, the third may include a comma. If the information from the database is too short for the pattern, for example a six-letter surname in a 15-letter pattern, then the rest of the pattern is padded out with spaces. If the information is too long, then the last five letters of

the name are ignored. The length of each of the macro parameters is controlled explicitly by the length of the pattern.

The Field list must contain one item for each pattern in the Format line, just as with normal reports. The item for the extra last pattern should be |R.

A complete example to define a multi-macro for the standard letter in figure 17.7 is given in figure 17.9. Notice how a comment is used to add the appropriate date to the letter. The 'Attn:' and the space before the name field are added to the macro parameter explicitly, by putting them within Half1, rather than extracting them from the database. 'Attn:' could just as easily be put into the VIEW macro definition.

When the report definition file is complete, press ESCAPE to return to ViewStore Command mode.

```
L Space 195    Indexed by entry
                          Report format de nition
Type (C=comment,H=header,M=margin,P=pagelength,R=record,S=subtotal,T=total)

T.Half1.......................................................................
R AA@@@@@@@@,<@@@@@@@@@@@@@@@@@@@@@@@@>,@@@@@@@@@@@@@@@,@@@@@@@@@@@@,@@@@
R ABAttn: @@@@@@@@@@@@@@@@@@,@@@@@@@@@@,@@@@
C Today's date?


Half2..........Field list..................
               ^C1,Street,Town,County,|R
               Name,Salutation,|R
               Todays date?
```

Figure 17.9. Multi-macro ViewStore report definition.

Printing a Multi-Macro Report

VSMACRO is a printer driver that first appeared in Acorn User, November 1987. It can be loaded by typing:

PRINTER   VSMACRO

In ViewStore Command mode. This printer driver should only be used to 'print' multi-macro reports. The effects with normal reports are unpredictable, though the database itself won't be damaged.

The database must also be loaded. After this, run the REPORT utility in the normal way, and answer the prompts as follows:

```
Use select file (N,Y)? Y
Screen or printer (S,P)? P
Use report format file (N,Y)? Y
Report filename? MMREPT
Send totals to linking file (N,Y)?
Subtotal field?
Single sheets (N,Y)?
Today's date? 21 June 1987
Macro filename? V.MMCALLS
```

This uses a report file called 'R.MMREPT'. The last two prompts are new.
The first of the two comes from the comment line in the report
definition, the second simply requests the name of the file to put the
macro calls in, in this case a file called 'V.MMCALLS'.

The information is then extracted from the database by the REPORT
utility, and the macro calls are put into a file. One macro or multi-
macro call is produced for every record in the database, or one for every
record in the selection file if the SELECT option was chosen.

If an error occurs during this process, or ESCAPE is pressed, then the
files are closed and the program returns to ViewStore Command mode.

Using the Macro File

The macro file can then be loaded into VIEW for checking, or merged
with the letter file containing the multi-macro definitions. The easiest
way to print the whole set of letters is to enter VIEW, load the correct
printer driver, and type:

```
PRINT <letter filename> <macro filename>
```

The letter file should contain only the definitions of the multi-macro
making up the standard letter, like the file shown in figure 17.7. The
macro file should contain the calls with parameters extracted from
ViewStore. You could use SCREEN instead of PRINT if just a quick
preview of the letters is required.

There is an important consequence of the VSMACRO method. The macro
parameters produced are of fixed length, the number of characters in
each field (@0, @1, @2, and so on) is controlled, and the parameter is
truncated or padded out with spaces as necessary. The standard MACRO
utility produces parameters that are of variable length, so a name could
be five or 50 characters long. VSMACRO makes them the same length as

the relevant pattern in the report definition. So if the macro parameters in figure 17.8 had been taken from a database with VSMACRO, they would have looked like figure 17.10. Notice that a couple of the parameters have been truncated because the data taken from the database exceeded the patterns in the report definition. Others have been padded out. Notice too the extra number at the end of each line - this is taken from the |R in the report definition.

```
AA  21.6.87 ,Moseley Constructi,Cucumber Road   ,<Keighley, W. York>,  17
AB  Attn: Installation Manager   ,faithfully  ,   17
AA  21.6.87 ,17 Whin Terrace    ,Thornthwaite   ,<Co. Durham      >,  18
AB  Attn: Joseph Addyman       ,sincerely   ,   18
```

Figure 17.10. Two VSMACRO calls to figure 17.7 multi-macro.

Advanced Use of VSMACRO

The padding spaces left at the end of the macro parameters can be removed if necessary. Load the macro file into VIEW and type:

```
CHANGE/ ,/,/
CHANGE/ >,/>,/
```

then resave the macros. Excess spaces are recognised because they precede a comma. Neither ordinary spaces by commas within angle-bracketed parameters, nor the leading spaces before numeric parameters are removed, as these occur in their usual place, after a comma. This is exactly what is required to keep everything neat.

The real function of the extra |R is to ensure this space stripping works properly on all current versions of VIEW. In fact, a comma and any single character would do, but using |R has a side effect; all the multi-macros are serial-numbered (up to 9999). The number is ignored by VIEW; it isn't printed and has no effect on the rest of the macro. But if standard letters should be serial numbered, then use this extra field - if a macro normally has four parameters, @0 to @3, then @4 is the serial number.

VSMACRO allows any highlights used in the field list of the report definition to be transferred into the VIEW macro file.

Using VSMACRO offers one more facility, the ability to join two or more pieces of information from separate database fields, to produce a single

VAV—S

macro parameter. The standard MACRO utility insists that one database field equals one macro parameter. But consider this Format line:

```
T.Half1....................................
R. AD<@@@@@@@@@@@@@@@@@@@@, @@@@@@@@@@@@@@@@@@>,@@@@
```

The macro AD has only one parameter (discount the extra one at the end for |R), because VIEW will treat the two patterns enclosed by angle brackets together as one. The patterns will be replaced by data from two database fields. A job title field and a company name field could be joined, for instance. The macro call might eventually be:

```
AD   <Sales Representative, British Zinc Co.>,    1
```

The space and the comma after 'Sales Representative' are derived explicitly from the Format line, like the 'Attn: ' in the figure 17.7 example. With joined fields like this, it is almost certainly necessary to remove the padding spaces, to keep the parameters neat.

Program Listing 17.1

```
 10 REM ViewStore macro spooler source
 20 REM by Graham Bell
 30 REM for B/B+/M/E + ViewStore/VIEW
 40 REM (C) Acorn User 1987
 50 :
 60 brkv = &202
 70 osfind = &FFCE
 80 osbput &FFD4
 90 osasci = &FFE3
100 osword = &FFF1
110 osbyte = &FFF4
120 :
130 DIM store &FF
140 FOR I% = 0 TO &FF
150 I%?store = 0
160 NEXT
170 offset = store - &400
180 :
190 FOR pass = 0 TO 2 STEP 2
200 P% = store
210 [
220 OPT pass
230 JMP output - offset
240 RTS
250 .last
260 BRK
270 .char
280 BRK
290 JMP close-offset
300 RTS
310 .nbrk
320 OPT FNequw(break-offset)
330 RTS
340 .handle
350 BRK
360 :
370 .text
380 OPT FNequs(" ? emanelif orcaM")
390 OPT FNequw(&0F0D)
400 :
410 .osblock
420 OPT FNequw(filename-offset)
430 OPT FNequb(&4FF-(filename-offset))
440 OPT FNequb(33)
450 OPT FNequb(126)
460 :
470 .openfile
480 LDY #&12
490 .message
```

```
 500 LDA text-offset,Y
 510 JSR osasci
 520 DEY
 530 BPL message
 540 LDA #0
 550 LDX #(osblock-offset)MOD256
 560 LDY #(osblock-offset)DIV256
 570 JSR osword
 580 BCC noescape
 590 LDA #&7C
 600 JSR osbtye
 610 BRK
 620 OPT FNequb(&80)
 630 OPT FNequs("Escape")
 640 BRK
 650 .noescape
 660 LDA #&80
 670 LDX #(filename-offset)MOD256
 680 LDY #(filename-offset)DIV256
 690 JSR osfind
 700 TAY
 710 BNE sethandle
 720 BRK
 730 OPT FNequb(&81)
 740 OPT FNequs("Can't open file")
 750 BRK
 760 :
 770 .close
 780 LDY handle-offset
 790 .closeall
 800 LDA #0
 810 JSR osfind
 820 .sethandle
 830 STA handle-offset
 840 .vector
 850 SEI
 860 LDA brkv
 870 LDX nbrk-offset
 880 STA nbrk-offset
 890 STX brkv
 900 LDA brkv+1
 910 LDX nbrk+1-offset
 920 STA nbrk+1-offset
 930 STX brkv+1
 940 CLI
 950 RTS
 960 :
 970 .break
 980 PHA
 990 TXA
1000 PHA
1010 TYA
1020 PHA
```

```
1030 LDY #0
1040 JSR closeall-offset
1050 PLA
1060 TAY
1070 PLA
1080 TAX
1090 PLA
1100 JMP (brkv)
1110 :
1120 .output
1130 STA char-offset
1140 TXA
1150 PHA
1160 TYA
1170 PHA
1180 LDY handle-offset
1190 BNE isopen
1200 LDA #&0D
1210 STA last-offset
1220 JSR openfile-offset
1230 .isopen
1240 LDA #&0D
1250 CMP last-offset
1260 BNE notend
1270 CMP char-offset
1280 BEQ return
1290 LDA #&80
1300 JSR osbput
1310 .notend
1320 LDA char-offset
1330 STA last-offset
1340 BPL print
1350 AND #&7F
1360 BEQ htone
1370 LDA #&01
1380 .htone
1390 ORA #&1C
1400 .print
1410 JSR osbput
1420 BIT char-offset
1430 BMI return
1440 JSR osasci
1450 .return
1460 PLA
1470 TAY
1480 PLA
1490 TAX
1500 LDA char-offset
1510 RTS
1520 :
1530 .filename
1540 ]
1550 NEXT pass
```

```
1560 :
1570 sum = 0
1580 FOR I% = 0 TO &FF
1590 sum = sum + I%?store
1600 NEXT
1610 IF sum <> &500B PRINT'"Checksum error - please check
     listing": END
1620 :
1630 PROCoscli("SAVE VSMACRO " + STR$~store + " +100 40C 400")
1640 END
1650 :
1660 DEF FNequb(byte)
1670 ?P% = byte
1680 P% = P% + 1
1690 = pass
1700 :
1710 DEF FNequw(word)
1720 ?P% = word MOD 256
1730 P%?1 = word DIV 256
1740 P% = P% + 2
1750 = pass
1760 :
1770 DEF FNequs(string$)
1780 $P% = string$
1790 P% = P% + LEN string$
1800 = pass
1810 :
1820 DEF PROCoscli(string$)
1830 DIM X% &FF
1840 Y% = X% DIV 256
1850 $X% = string$
1860 CALL &FFF7
1870 ENDPROC
```

18 : Putting on a Show
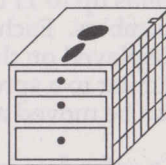
ViewPlot

ViewPlot is a utility for displaying numerical data fromViewSheet or
ViewStore in a graphical form. The package consists of a suite of BASIC
and machine code programs on disc, plus a function key strip. Before
starting to use ViewPlot, make a back-up of the ViewPlot disc. Use the
copy and keep the original in a safe place. With a 40-track drive, the
back-up can be made with the *BACKUP command. An 80-track drive
requires the use of the *CONFIG utility on the ViewPlot disc. Align the
key strip above the function keys.

ViewPlot works with DFS, ADFS or the Network Filing System. On
Econet, consult the network manager about making ViewPlot available
on the fileserver.

ViewPlot can be started up by auto-booting the disc using SHIFT–BREAK.
If all is well, this displays a six-item menu showing the available
options for entering data and chart plotting. Choices can be made by
pressing the number key associated with each option. It is best to keep
the ViewPlot program disc in the drive, because ViewPlot uses a system
of *overlays*; parts of the ViewPlot program are kept on the disc and only
loaded in when they are needed.

Entering Data into ViewPlot

All data plotted with ViewPlot must be put into a ViewPlot data file first.
This data can be typed in manually, or it can be taken from a ViewSheet
link file, or from a spooled ASCII file. This is done using main menu
option one.

The simplest way is to enter the data manually. The data editor
presents a screen display showing a status area at the top, plus two
data entry windows. The main, lower window is divided into three
columns - Labels, X and Y. The Labels (or L), column can contain text

items up to 11 characters long, but the X and Y columns can contain only numbers. Each column is divided into rows, and the row numbers are displayed on the left of the screen. The working of the screen display is similar to a spreadsheet. A cell cursor is shown, initially in cell L1, but it can be moved about with the cursor keys.

To enter data into a cell, just type it in. As it is typed, it appears in the status area at the top of the screen, where there is a flashing cursor. But when RETURN is pressed, the data is transferred to the cell, and the cell cursor jumps automatically to the next cell. As an item is added to the L column, matching data appears in the X and Y column. This is initially set to zero, but any number can be typed into these columns, with any number of decimal places. However, data in the X and Y columns is always shown with exactly two decimal places.

To edit an item, simply return the cell cursor to the cell, and retype it correctly. To delete it altogether and leave a blank cell, type a space then press RETURN. The function keys f6 and f7 can also be used to edit the data. f6 puts in a new blank row at the current cell cursor, allowing a forgotten item to be inserted, and f7 deletes the current row and closes the gap by moving succeeding data items up.

Function key f0 controls auto-entry. When it is on, 'Auto-entry' is shown in the status area, and pressing RETURN sends the cell cursor to the next cell. Switching auto-entry off leaves the cell cursor static, and the UP, LEFT, RIGHT and DOWN keys need to be used to position the cell cursor. The cursor can also be moved to a particular row by pressing f8, then typing in the row number. So:

    f8 GOTO 12

moves the cursor to cell L12. Data can be entered on up to 100 rows, although only 15 rows are shown at any one time.

Pressing f1 allows any star command to be used. A blank screen is shown, with a '*' prompt. For example, typing:

    f1 * CAT

shows a catalogue of the files on the disc. To return to the data editor screen, just press RETURN. Note that using some commands, *BACKUP or *COPY for example, are likely to corrupt the ViewPlot program and

crash the computer, so ensure only 'safe' operating system commands are used. Unsafe commands are listed in Appendix One.

The TAB key moves the cell cursor to the upper window, and a second press of TAB returns it to the lower window again. In the upper window, various text items can be typed in. This is used to annotate the data when the final graph is displayed. Each item can be up to 11 characters long.

After the data is complete and correct, it can be saved in a ViewPlot data file. To save the file, press f3, and type the filename at the prompt. In a similar manner, data files can be reloaded and edited, using f2. With DFS or ADFS, the filename can include the drive number, so that the data need not be kept on the ViewPlot program disc. To save the data on the disc in drive one, in a file called 'D.NUMBERS', type in:

```
f3
File to save:1.D.NUMBERS
```

Using directory D for data files is a good idea because ViewPlot also uses 'format files'. When the data has been saved, return to the menu by pressing ESCAPE - because this can lose any unsaved data, it must be confirmed with 'Y'.

Types of Graph

Figure 18.1 shows a data editor screen, with each part of the display filled in. The same data is shown again in figures 18.2, 18.3 and 18.4, but in the final graph form. Any of the graphs can be produced with main menu option three.

Option three produces a series of prompts to select the type of graph display required:

```
Enter screen mode (4 or 5)? 4
Do you want default colours? Y
Do you want patterns? Y
Do you want a border? Y
Data/format file name? <data filename>
```

With a micro incorporating extra shadow memory, modes 0, 1 or 2 may also be available. A border is simply a line round the edge of the screen, and is almost always appropriate. The data/format filename is the data file saved from the data editor.

The colour and pattern prompts refer to the use of special files containing screen textures that can be used to differentiate sectors of a pie chart, for example. ViewPlot has a built-in set of textures, so no new special file is needed. However, ViewPlot has a pattern editor, reached from the main menu with option five, that allows new colours or textures to be specified and incorporated into graphs.

Once the name of the data file is entered, ViewPlot asks which type of display is required:

```
Do you want a Bar, Line or Pie graph?
```

Press 'B', 'L' or 'P', and the screen clears and the desired graph is plotted. The three figures illustrate the three types of display ViewPlot generates. The important point to notice is that none of the charts contain all the information - each displays only a part of the data. Note also where the text is placed on each type of plot; ViewPlot allows no control over this, so make the best of the titles and axis labels that can be incorporated.

```
        Edit Data Set              Length:  8

                         Auto Entry

        Title        MAIN TITLE
        Labels name  LABELS NAME
        X-axis name  X-AXIS NAME
        Y-axis name  Y-AXIS NAME

        Labels       X            Y

     1  A            8.00         1.00
     2  B            7.00         2.00
     3  C            6.00         4.00
     4  D            5.00         8.00
     5  E            4.00         16.00
     6  F            3.00         32.00
     7  G            2.00         64.00
     8  H            1.00         128.00
     9
    10
    11
    12
    13
    14
    15
```

Figure 18.1. Completed ViewPlot data set.
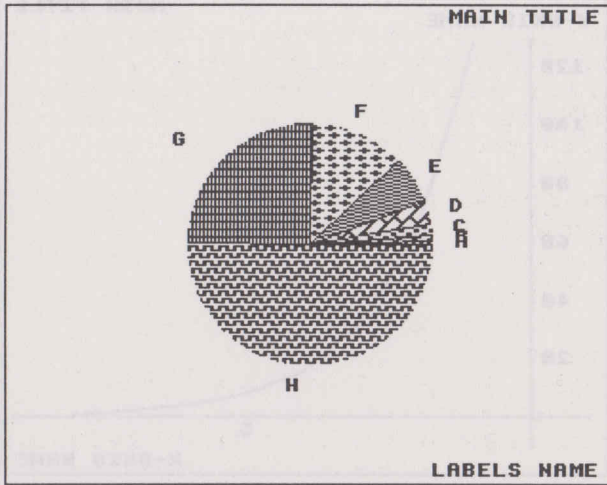
Figure 18.2 Pie chart illustrating fig 18.1. data
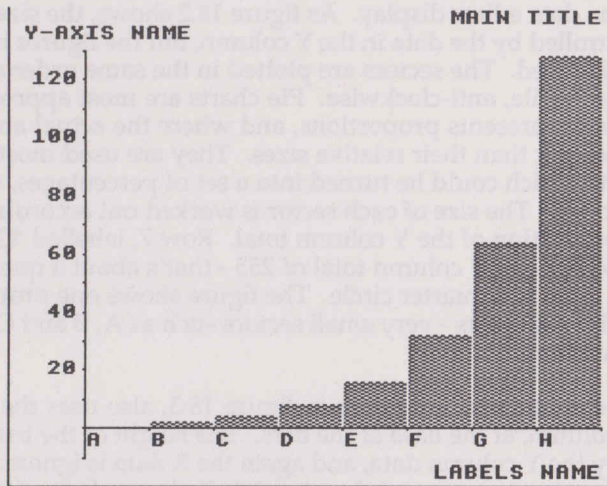
Figure 18.3 Bar chart illustrating fig 18.1. data
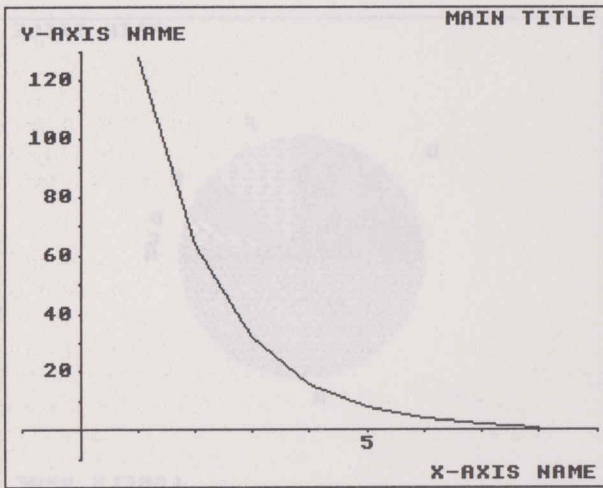
Figure 18.4 Line graph using fig 18.1. data

Pie charts show sectors, each labelled with the text taken from the L column of the data editor display. As figure 18.2 shows, the size of the sector is controlled by the data in the Y column, but the figures in the X column are ignored. The sectors are plotted in the same order as the rows in the data file, anti-clockwise. Pie charts are most appropriate when the data represents proportions, and where the actual amounts are less important than their relative sizes. They are used most to illustrate data which could be turned into a set of percentages, all adding up to 100. The size of each sector is worked out according to that row's proportion of the Y column total. Row 7, labelled 'G', represents 64 out of a Y column total of 255 - that's about a quarter, so the sector is close to a quarter circle. The figure shows one problem that can occur with pie charts - very small sectors such as A, B and C make the labels overlap.

A *bar chart*, or *histogram*, as shown in figure 18.3, also uses the text from the L column, at the base of the bars. The height of the bars is controlled by the Y column data, and again the X data is ignored. The bars are drawn in row order, and each is labelled at its base. ViewPlot makes an intelligent choice about the total height of the Y axis, searching for the largest item in the Y column. It also automatically marks the round numbers along the axis. The width of the bars limits

the length of the labels - in the example, only four characters could be displayed at the base of each bar. With more than eight rows of data, there would be narrower bars, so the labels would have to be shorter. Bar charts are useful for many types of data, particularly where it is aggregated into groups of equal importance or where non-numerical comparisons are being made - each bar could be a monthly total labelled JAN–DEC, or could illustrate journey miles on different types of transport, for example.

The *line graph* in figure 18.4 shows points linked by straight lines. This is the only type of plot that takes notice of both the X and Y columns of the data, though the text in the L column is completely ignored in the line graph. The exact position of each of the points is controlled by the X and Y data. In the usual Cartesian manner, X gives the position left to right along the x-axis, and Y gives the height of the point up the y-axis. The points are linked by lines in the row order they occur in the data file. As with the histogram ViewPlot makes automatic choices about the length, subdivision and labelling of the axes. Line graphs like this are usually used to show how one numerical quantity depends upon another, say how the quoted cost of a component depends upon the order size - order more and the cost of each one falls.

It's important to contrast figures 18.3 and 18.4, which apparently show contradictory trends in the same data. This arises because the bars in figure 18.3 are plotted in row order, and the low-numbered rows have low Y values - remember the X values are ignored. The low X values in figure 18.4 are associated with high Y values, and high X values with low Y values. So it is important to work out what the Y values are being plotted against. Do they depend on the row, or are the X values important?

Because each chart needs only some of the data, parts of the data can safely be left blank. There is no need to fill in the L column of the data if only a line graph is to be plotted.

ViewPlot and Printers

ViewPlot charts can be printed out by pressing SCREEN PRINT (function key f9), after the chart has been displayed on screen. However, the screen print program works only on Epson FX-compatible printers, and only if they are set to give automatic line feeds. Luckily, there is also a facility to save the screen to disc. Key f8 is labelled SCREEN DUMP on the

function key strip, but in fact it puts the screen display into a file on disc called IMAGE. This can be reloaded later using the *LOADUMP utility on the ViewPlot disc. If a screen print program is available for the attached printer, or if there is a ROM-based screen print utility such as Acorn User's UserDump fitted, then the following BASIC program can be used to reload the saved screen and print it:

```
10 REM reload and print screen
20 :
30 *LOADUMP IMAGE
40 *SDUMP
```

This assumes that the screen print program or command is called *SDUMP; if the command is different from *SDUMP, or if extra parameters are needed, then the program can be amended. The important point is that *LOADUMP reloads the screen and restores the mode and colours that were in use when the screen display were saved.

An alternative way of doing the same thing is to put a new file PRTDUMP on the ViewPlot disc. Most existing screen print programs use a section of memory that does not conflict with ViewPlot, often at &900 - a selection of such programs have been published in *Acorn User* magazine, for example in the September 1986 and May 1987 issues. If a program known to work with the printer is put on the disc and called 'PRTDUMP', then ViewPlot will use this automatically. Remember to *RENAME the old 'PRTDUMP' to something else, perhaps 'OLDDUMP' before copying the new program onto the disc.

Many printed screens show circles as ellipses. Search for a proportional dump program. This is a screen printing program that makes sure that circles on screen, such as pie charts, really do print as circles!

Automatic Data Entry

ViewPlot can be used manually, but it is more sensible to use it to display numerical data taken from either ViewSheet or ViewStore. ViewPlot can read ViewSheet link files or ordinary ASCII text files. Remember that ViewStore can create link files too, using the LINK utility.

The procedure to read from a link file is probably the simplest. Graphing consolidated data is the usual need for ViewPlot, perhaps so the chart can be inserted into a report document. Chapter 15 described a link file V.VS3 used to consolidate results from several independent

work groups in a school project. This file has four columns, and as many rows as there are groups. Each group's results are written into the link file, using one row for each of the different groups. Only two results are taken from each group, and are put in the first two of the four columns. The following procedure reads the results in column two straight into ViewPlot:

```
f4
Link file: V.VS3
The file is 4 columns by 8 rows
Do you wish to read a column or a row?
Enter (C/R) C
Which column do you wish to read from?
Enter number (1-4) 2
Start row
Enter number (1-8) 1
Finish row
Enter number (1-8) 8
Do you wish to read this as X or Y data?
Enter (X/Y) Y
```

The second column of the link file, in this case containing the co-efficient of variation of the pebble sizes measured by the group, is read into the Y column of the data editor. In a similar way, the average pebble sizes from column one of the link file can be read into the X column. Save the ViewPlot data file in the normal way, and it can then be used to construct graphs.

By varying the link file row or column number, and the start and stop column or row, any part of the link file can be read into the ViewPlot. But note that the first data read is always put into row one of the data editor. If more than one part of the link file has to be read into the X column say, then the second lot will overwrite the first. The technique here is to read the second lot into the data editor first, then use f6 to insert enough blank lines above it to leave room for the first lot of data.

To draw graphs of data where a link file does not already exist is fairly simple. Somewhere in a model, there is usually an area where the 'results' are calculated. For example, this may be a row where the sums of various columns are totalled. These results have to be written to a link file. The way to do this is to create a *map* of the link file on the spreadsheet. Figure 18.5 shows a typical map. The idea is to duplicate the results area of the model, and write it to the link file. The formula in cell D16, shown means that the contents of cell D11 should be written to file V.VS4. The co-ordinates within the file used to store this result is

dictated by the 'ROW-A14,COL-A15' part of the formula, and in this case equates to link file cell (2,3). A14 and A15 contain the formulae ROW and COL respectively. Of course, the map formulae all refer to matching cells in the results area of the model, and can be set up fairly easily using replication.

```
LA  SLOT=D16

CONTENTS=WRITE(4,ROW-A14,COL-A15,D11)

        .......A.......B.......C.......D.......E
.....9          group1  group2  group3  group4
.....10 result1    41      78      65      27
.....11 result2    38      74      64      22
.....12 result3    39      77      66      31
.....13
.....14     14
.....15      1     41      78      65      27
.....16            38      74      64      22
.....17            39      77      66      31
```

Figure 18.5. Model with a 'map' to create link file.

Figure 18.5 also shows that it is sometimes necessary to transpose rows and columns in a map. Cell D16 is in the third column and second row of the map, and yet its result is put in column 2, row 3 of the link file.

This transposition is needed because the results for each group lie within a column - grouped results within ViewPlot must lie along a row. Within the data editor, the final results may look like this:

```
LABELS      X       Y

group1      41      38
group2      78      74
group3      65      64
group4      27      22
```

Note that only columns one and two have been read into the editor, and that the labels have been added by hand. Transposition can be done by careful answering of the prompts presented while reading the link file into the editor. It is a good idea to jot down beforehand the way the data should finally look. Even a rough note can help to show when a mistake has been made.

To read a spooled text file is equally straightforward; press f5, then type the name of the file to read. ViewPlot then asks for the data format that describes the way that the data is laid out in the text file, and how it should be read into the ViewPlot data editor:

1 column of data - read into	L column	4
	X column	2
	Y column	1
2 columns of data - read into	L and X	6
	L and Y	5
	X and Y	3
3 columns of data - read into	L, X, Y	7

Just press the number corresponding to the data format required. Easy!
It is producing the text file that is a little bit more difficult, but ViewPlot
supplies a utility program called *PREPARE to automate part of the
process of text file creation. The general procedure is to set a printer
window around the area of the ViewSheet model to be transferred to
ViewPlot. Altering the printer windows is described in Chapters Four
and Five. The window should enclose either one, two or three columns
of data, and can be up to 100 rows deep. The top and side margins
should be switched off with the T and S options, for example like this:

```
Wi TopL  BotR  Pos Cw  Bw Fmt   Opt
P0 C4    E16       7    7  D1RM  TS
```

It is also worth checking that the format gives sufficient precision, and
the column width is wide enough to display the numbers and and labels
correctly. Finally, make sure negative numbers are not shown
surrounded by brackets, and switch any other printer windows off by
using the O option. To use *PREPARE, set up the printer window and
return to Command mode, then type:

*PREPARE <filename>

This automatically spools a special text version of part of the model
into a file. This spooled file can then be read into ViewPlot using f5. In
principle, other text-only files can be read too. For example, text could
be prepared or edited in VIEW, then read into ViewPlot, or report files
from ViewStore could be spooled using the ASCII printer driver
described in Chapter Six. Text could even be taken from more than one
ViewSheet printer window, providing they are set up correctly.
However, because of the way the data editor reads the file, there are a
couple of caveats! First, there can't be any spaces within a column, so
labels cannot consist of more than one word. A space signals the start
of the next column, so it would cause the rest of that row to be
misinterpreted. If a label with more than one word must be included,
then it is best to link the words with underline characters rather than

spaces. Second, ViewPlot can't read ordinary text files properly because it ignores the first and last lines of the text file, and also the first character of each line. This doesn't affect *PREPARE, because it sets the first line to '=>SC' and the last to '=>*SPOOL', and each line is prefaced by a space. To make the ViewPlot data editor better able to read ordinary text files prepared in VIEW, or spooled output from other programs, enter BASIC, and insert a copy of the ViewPlot disc.

Load the program 'V--E', and type in the following changes:

```
3250 IFEOF#a%GOTO3390
3340 PROCC:I%=I%+1:UNTIL(I%>I)OREOF#a%
3480  %=0:ENDPROC
*RENAME V__E OLDV__E
SAVE "V__E"
```

Before renaming 'V__E', the file may have to be unlocked with the *ACCESS command.

The new version of the data editor should be capable of reading almost all text files properly, except those made using *PREPARE. Creating normal text file versions of spreadsheets for the modified data editor is similar to using *PREPARE. Set up the printer window as before, load the ASCII printer driver, and use the PRINT command. Carefully designed ViewStore reports could be read in, if a text file version was previously produced with the ASCII driver.

Multiple Data Sets

ViewPlot has a facility for displaying more than one graph on the screen at the same time, and also is able to plot more than one set of figures on the same graph. This is done via the main menu option two, 'Select Charts'. The sets of data must first be saved in separate data files using the data editor, option one.

The format editor presents a three window display as shown in figure 18.6 - a window to specify which data files should be plotted, a window to put titles for the charts in, and a third window specifying a 'chain file'. This last allows a series of format files to be set up, which can then be used in a carousel display.

The display moves on to the next format file in the chain when the space bar is pressed.

```
Format File                      Charts: 2

                                 Auto Entry
           Data sets    1 2

       1  D.DATA        B L
       2  D.DATA2       B L
       3
       4
       5
       6
       7
       8
       9
      10

          Chart titles   Chain

       1  BAR GRAPH 1
       2  LINE GRAPH 2
```

Figure 18.6. Format editor screen.

The format editor works in much the same way as the data editor; the cursor keys can be used to move around each window, and TAB switches between windows.

In the main window, the names of any data files should be specified. In this case, two sets of data are being used. The numbered columns can be used to say what type of graph is to be plotted with each data set. The number of useable columns can be changed with the NUMBER OF CHARTS function - press SHIFT–f8 and it changes to two, then to four, then returns to one. In each column, type 'B' for a bar chart, 'L' for a line graph, or 'P' for a pie chart. A single data set can be plotted on more than one of the charts if necessary, as shown in the figure. The format requests two charts, the first a histogram, the second a line graph, each showing both the data sets. Figure 18.7 shows this chart plotted in mode 0. However, the format could easily be set so that the two data sets were compared on separate line graphs, for example:

```
      Data sets    1 2

   1  D.DATA          L
   2  D.DATA2         L
```

It is possible to have one, two or four charts plotted on the same screen. Each can display up to 10 data sets, except for pie charts which show only one data set. Labels for axes are taken from the first set of data for each chart, and the axes are intelligently labelled according to the widest range of values in any of the data sets.
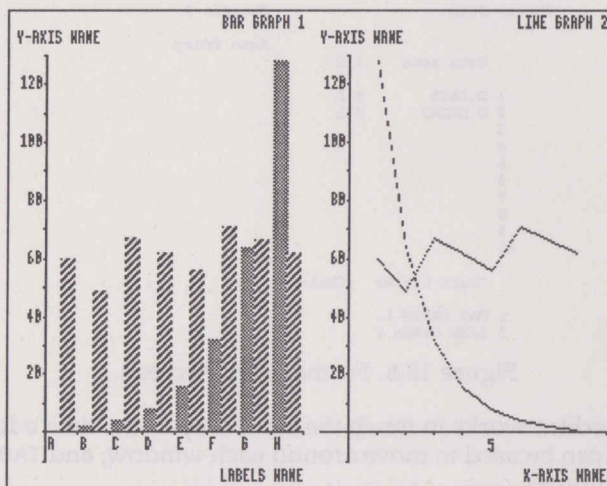
Figure 18.7. Multiple chart using figure 18.6. format

ViewChart

ViewPlot has a rival: Acorn User's ViewChart package. In most respects, ViewChart and ViewPlot are very similar - the types of graphs they produce can be used interchangeably in most circumstances. However, there are differences in approach between the packages, so one may be more suited to a particular application than the other.

ViewPlot offers both patterns and colour on screen. On a standard BBC model B micro, it can be used in mode 5, and graphs can be displayed in four colours. Alternatively, differing patterns and textures can be used to differentiate parts of a chart even in a two-colour mode (mode 4 on a model B). ViewPlot only uses the high-resolution screen modes with a BBC B+, a 6502 second processor or with a Master series micro. In contrast, ViewChart always uses mode 0, even on a standard model B. This means that there are no colours on the screen, but it does allow the graphs to be drawn with the maximum detail.

ViewChart only allows one bar or line graph to be drawn on screen, whereas ViewPlot can display up to four different graphs at once. For this reason, ViewPlot is better at showing variation among data sets.

However, the labelling of charts is much more flexible in ViewChart - text can be placed anywhere on screen, whereas ViewPlot dictates automatically how and where to label axes and graphs.

Overall, this boils down to a choice - ViewChart aims at producing the best printed output, even on a model B. ViewPlot offers attractive, coloured screen displays, but can only produce high-resolution printed graphs with a B+ or Master series micro, or a 6502 second processor.

ViewChart is limited to only 25 bars in a bar chart, or 40 points in a line graph. Pies may have only five sectors, although there may be up to five pies on a single screen. These are more severe limitations than the 100 data values that ViewPlot can use, but are inevitable given the memory constraints of working in mode 0.

ViewChart can use data typed in by hand, and like ViewPlot, it can read ViewSheet link files. These can easily be created using a map of the results area of the model. ViewChart cannot, however, read spooled text files. In recompense, it can read data directly from VIEW or ViewStore macro files; this allows text labels as well as numbers to be read in. In VIEW, the macros should be of the form:

```
F I.......*.......*.......*.......*.......*.<
XX text label,12
```

The text and number correspond logically to the L and Y columns of the ViewPlot data editor. In fact, up to five numbers can follow the label, and in this way ViewChart can keep multiple data sets in one file rather than using separate data files plus a format file as ViewPlot does.

19 : OverView
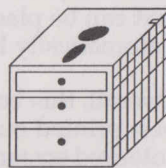
The main OverView software is supplied in a single cartridge for the Master 128 containing ViewStore, ViewSpell, and the extra OverView utilities such as the wide screen system, and *HELP information on the whole VIEW family. A supplementary ADFS-format 5.25-inch disc contains the ViewIndex and ViewPlot software, as well as the ViewStore utilities plus ViewSpell dictionary. In order to install OverView, make sure the computer is switched off, then insert the cartridge into the slot behind the numeric keypad on the Master 128. Now switch on the computer. Typing *ROMS should show ViewStore and ViewSpell occupying either positions zero and one or alternatively two and three, depending which cartridge slot the OverView cartridge is in. It's best to make a back-up of the OverView disc to use and keep the original in a safe place.

OverView is also available for the Master Compact as part of the Compact Professional package. This version also includes ViewSheet. All the software is supplied on a single ADFS 3.25-inch disc, and ViewStore, ViewSheet and ViewSpell can be loaded into the computer's sideways RAM banks. Again, don't use the original disc, make a back-up, and keep the original safe.

Using OverView

In use, the OverView versions of the VIEW family software are identical to the original versions in most respects. The only differences to be aware of are that the OverView disc is in ADFS format, and it is probably best if ADFS is used throughout. This means that when using ViewStore, the filename prefixes can be longer. For example, the utilities are on the disc in directory '&.STORE.U'. Either select the 'STORE' directory with:

    *DIR &.STORE

or set up the prefix to take the directory structure into account:

```
PREFIX U :0.&.STORE.
```

Other directories on the disc are called '&.SPELL', '&.PLOT', '&.INDEX' and '&.PDG'. Like ViewStore, ViewSpell allows prefixes, but for the other applications use *DIR to select the correct directory as above. On a working copy of the OverView disc, a series of EXEC files can be set up in a library to automatically select the right directory and run each application. For example:

```
>*CDIR &.LIBRARY
>*BUILD &.LIB*.PDG
    1 *DIR &.PDG
    2 *BASIC
    3 CHAIN "PDRGEN"
    4 ESCAPE
```

builds a command file called PDG. Typing *PDG has the same effect of *EXEC PDG. The commands will be obeyed as if they had been typed by hand to run the Printer Driver Generator. In this way, swapping between applications is much more convenient.

The Keeper

This OverView utility is meant to help you switch between VIEW, ViewSheet, ViewStore and ViewSpell. To see how it works, first make sure there is a directory on the current disc called '&.+'. Try:

```
*DIR
*CAT +
```

and if a 'Not found' error message occurs, then type in:

```
*CDIR +
```

to create the Keeper directory. Switch the Keeper utility on with the following command:

```
*KEEP ON &.+
```

Now enter one of the VIEW family applications, say ViewSheet, in the normal manner. Notice that the disc drive whirrs briefly as ViewSheet starts up. Create a small spreadsheet, perhaps even set up a couple of windows, then press ESCAPE. Don't save the model, but switch

immediately to another application, say VIEW. Again the disc drive will whirr. Write a little letter, then go back to ViewSheet. Again there should be a pause while the disc drive whirrs. This time, ViewSheet will be restored to the exact point where you left off, with the spreadsheet in memory. At this point, check the disc:

```
*CAT &.+
```

There should be six files as follows:

AVIEW	ASheet	BVIEW
BSheet	CVIEW	CSheet

These files contain the VIEW and ViewSheet contexts; that is the state of the micro when VIEW or ViewSheet were last left. So the three VIEW files contain all the necessary information to allow the Keeper to pick up the threads of the demonstration letter written before returning to ViewSheet. It used the three Sheet files to restore the demonstration model when ViewSheet was re-entered.

The Keeper will save 12 files in total, three for each of the major VIEW family applications. So there is no need to save and then reload all work repeatedly when swapping between members of the family. The keeper will even save the current work when leaving the family altogether, say on typing *BASIC. The latest context is restored when a family application is next started up.

The Keeper always stores its context files in the '&.+' directory. Of course, the + directory was created specially for the Keeper. If *DIR is used to select any other directory, then the Keeper will always be able to find the right directory, because the prefix &.+ includes & to specify that the + directory is only one level away from the root directory. A different prefix can be added to the filenames used by the Keeper by using the command:

```
*KEEP ON &.C.
```

when switching on the Keeper. For example, the prefix '&.C.' ensures that the keeper will always use the '&.C' directory. In a system with two disc drives, the drive number can be part of the prefix too.

The Keeper doesn't end the necessity to save all finished work. It keeps only the current document or model. In VIEW or ViewSheet, when a piece of work is completed, it should still be saved in the usual manner.

Before switching off the micro, make sure that you save the current document or model. If necessary, the current context can be saved too, by switching to another application immediately before switching off: for example, switch to BASIC. Switching on the next time, the old context can be restored by starting whatever VIEW application was being used before switching off.

Advanced Use of the Keeper

With a Master 128 instead of keeping context files on disc, the Keeper can store them temporarily in sideways RAM. There are normally four banks of sideways RAM available in a Master 128, sufficient to store two sets of context files. Keeping contexts in sideways RAM is much quicker than keeping them on disc, and switching between VIEW applications doesn't involve any disc access. The Keeper will automatically store the first two sets of files in RAM, if it is activated by entering:

```
*KEEP RAM &.+.
```

Any subsequent contexts will be stored on disc as usual, using the prefix if one is specified. It's vital with contexts in RAM that the command:

```
*KEEP OFF
```

is used before switching the computer off. This transfers any contexts held in RAM to disc, so they are not lost. Data can't be kept in sideways RAM when the computer is switched off.

A Master Compact (except the Compact Professional) can't use sideways RAM with the Keeper, because some of the four RAM banks are used to store the ViewSheet, ViewStore and ViewSpell ROM images themselves.

Similarly with the Master 128, *KEEP RAM should not be used if any ROM images have been loaded with the *SRLOAD command, or if the sideways RAM is being used to store data, or if the internal hardware links L18 or L19 are set to allow ROMs to be plugged into the first and third sideways ROM sockets behind the cartridge slots.

If no prefix is specified in the *KEEP command, then the Keeper normally uses the '+' directory, but this is of little use. If *DIR is used to change directories away from the root directory $ (or &), then the Keeper won't find the '+' directory. It is always best to specify that '+' is a sub-directory of the root directory. However, the '+' can be changed if necessary, as it is part of the prefix. In contrast, data files in ViewStore are kept in directory 'D.'. A prefix like 'STORE.' means they are kept in directory 'STORE.D'. The D is not part of the prefix, and is always added on at the end. The current Keeper prefix can always be seen by typing the following:

```
*KEEP
```

A problem occurs with the Keeper on a Master Turbo. Very long VIEW documents can be too long to keep. If there more than about 4500 words in a document, then a 'Too much to keep' error message may be displayed. In this case, save the text as normal using the SAVE command. It will have to be loaded again by hand when it is needed next. The Keeper prevents exit from an application until a context is saved successfully, so when the text is saved, the command:

```
*KEEP  QUIET
```

should be used to switch the Keeper off. To enter another VIEW family application and restore the context, on no account must it be entered directly. First, switch to BASIC with *BASIC, then re-enable the Keeper with *KEEP ON or *KEEP RAM, then use *STORE, *SHEET or *SPELL.

Remember contexts saved while using a Master Turbo can't be reused without it. The exact hardware in use makes each context unique - even changing a ROM can make old context files unusable.

The Wide Screen

The *WIDE command can be used to increase the number of characters across the screen in any two-colour mode. It allows 106 characters in modes 0 and 3, or 53 in modes 4 or 6. It also works as expected in shadow modes 128, 131, 132 and 134, but not in any of the four or 16 colour modes. Fifty-three characters is a useful width for wordprocessing on a television, as 80-column text can be too indistinct to read easily. A good monochrome monitor is essential for 106-column modes - or example with large ViewSheet spreadsheets.

*WIDE ON switches the wide screen on, though it doesn't take effect until the next mode change. Conversely, *WIDE OFF switches it off at the next mode change. All the VIEW family make use of the wide screen if it is on - for example ViewStore displays the first 105 characters of each record across the screen in mode.

The wide screen does make displaying text more sluggish. Scrolling through a VIEW document or across a ViewSheet spreadsheet becomes noticeably slower than normal.

Reading Contents Files

The two OverView utilities *READ and *RC, allow text to be read into ViewSheet. *READ is for reading formatted text from VIEW or possibly other applications. *RC is for reading *contents* files, for reading part of one ViewSheet model into another.

A contents file is created like this. After loading the model into ViewSheet, type the following commands:

```
PRINTER ASCII
PC
```

This makes use of the ASCII printer driver described in Chapter Six. Alternatively, the VSXFER version could be used. The PC command is described in Chapter 13. As the contents file is created, the computer prompts for a name to call the file, say 'CONTA'. This contents file is purely text; it can be loaded into VIEW, as shown in figure 19.1.

```
F I.......*.......*.......*.......*.......*.<
   H4 10
   I4 12
   J4 14
   B6 11.1
   D6 B6-AVERAGE(B6B14)
   E6 D6^2
   G6 IF(B6>=G2,IF(B6<G4,1,0),0)
   H6 IF(B6>=H2,IF(B6<H4,1,0),0)
   I6 IF(B6>=I2,IF(B6<I4,1,0),0)
   J6 IF(B6>=J2,IF(B6<J4,1,0),0)
   B7 12.3
   D7 B7-AVERAGE(B6B14)
   E7 D7^2
   G7 IF(B7>=G2,IF(B7<G4,1,0),0)
   H7 IF(B7>=H2,IF(B7<H4,1,0),0)
   I7 IF(B7>=I2,IF(B7<I4,1,0),0)
   J7 IF(B7>=J2,IF(B7<J4,1,0),0)
```

Figure 19.1. Contents file from the BEACH model in VIEW.

An equivalent contents file, 'CONTB', can be created like this:

```
*FX 5 0
*SPOOL CONTB
PC
*SPOOL
*FX 5 1
```

The *FX 5 commands prevent the PC command actually printing all the cell contents from the spreadsheet. For serial or network printers, *FX 5,2 or *FX 5,4 should be used. *SPOOL CONTB ensures that the contents information is recorded in the file. The only difference between the two methods is that the CONTA contains only the contents, and it can be loaded into VIEW with the LOAD command. CONTB contains additional text (in fact '=>PC RETURN' and '=>*SPOOL RETURN'), plus a few control codes. These control codes mean it can only be loaded into VIEW using the READ command.

The *RC command can be used to read either type of contents file into another spreadsheet. For example, it's useful to copy a block of data from one sheet to another. This might happen if there is one set of data and two separate ways of analysing it. Type the data on to a mask for the first analysis, then copy it into the second mask file.

Start off by loading the first model from which information is to be taken. If only part of the model is to be transferred, then delete all the other cells by replicating blank cells over the unwanted areas. This ensures that any cell references remain correct within the transferred area. Next, make a contents file. Now load the second spreadsheet, and ensure that there is a blank area into which the contents file can be read. Clearly, if data comes from cell G6 on the first model, it will be read into cell G6 on the new model. Any existing data in cell G6 will be lost. The INSERT ROW and INSERT COLUMN functions can be used to create space if necessary. When there is a clear area for the data, in ViewSheet Command mode, type in:

```
*RC <filename>
```

giving the name of the contents file.

Of course, PC and *RC do not transfer attributes like the number format of a cell, so the transferred data may not immediately look like that in the original model. The process is much quicker if recalculation is set to manual (using function key SHIFT–f0), before using the *RC command.

Reading Text

OverView's *READ command allows plain text to be read into a spreadsheet. For example, look at figure 19.2 - this shows some text in VIEW. This shows some text in VIEW, formatted so that TAB characters separate each of the columns. This text should be saved in the usual way, using either the SAVE or WRITE commands. There should be no edit commands or rulers splitting up the text.

```
F I.......*.......*.......*.......*.......*.<
  Stock levels

     JAN     FEB     MAR     APR     MAY
     1       2       3       4       5
     6       7               9       10
     11      12      13      14      15
********************************************
```

Figure 19.2. Formatted VIEW text for transfer to ViewSheet.

Once the text is in a file, it can be transferred to ViewSheet. Start ViewSheet up as normal, and if the text is to be added to an existing model, load that too. As with the *RC command, ensure that there is a blank space on the spreadsheet to place the text, otherwise existing data will get overwritten. Switch recalculation mode to manual (key SHIFT–f0 from the Sheet screen) to speed up reading.

From ViewSheet Command mode, to read a text file, enter:

*READ < lename> <cell> T

giving the name of the text file to read. The cell reference specifies where on the model the text should be placed. If the file in figure 19.2 is called 'MODLTEX', then figure 19.3 shows the effect of the command:

*READ MODLTEX C2 T

Notice that each column of the text is placed in a separate cell. TAB characters in the text separate cells in the model, and the two adjacent TABs after '7' create a blank cell on the sheet. Each line of the text is placed on a separate row of the sheet too. The first line begins at the correct cell, C2, and each new row begins below this.

Another important point is that the text 'Stock levels' contains a SPACE character, not a TAB, so it is all placed in a single cell. Spaces are not taken as separating items in the text. This could be changed, using the following command:

```
*READ MODLTEX C2
```

as before, but without the final 'T'. This would place the text 'Stock' in
C2 and 'levels' in cell D2, as either TAB or a SPACE would be taken as
separating cells. But although two TAB characters can be used to create
a blank cell, two space characters don't. A group of spaces has the same
effect as a single space.

```
VM   SLOT=G5
CONTENTS=15


0        .......A.......B.......C.......D.......E.......F.......G
......1
......2              Stock 1
......3
......4              JAN      FEB      MAR      APR      MAY
......5               1        2        3        4        5
......6               6        7                 9       10
......7              11       12       13       14 >      15
......8
......9
```

Figure 19.3. Text file after transfer to ViewSheet.

*READ is invaluable for transferring any data that may be regularly
updated into a spreadsheet. For example, updating the price list in the
INVOICE model described in Chapter 15 is much simpler using VIEW. The
completed list, which is shown in ViewSheet in figure 15.4, may then be
read into the model at the correct location using *READ. It is not useful
transferring any material containing a cell reference, as this will
doubtless be wrong if it is read to a different location on the model.

Abbreviated Help

OverView contains terse *HELP text for all four main VIEW family
utilities. This can be seen by typing:

```
*HELP SPELL
```

for example. The text for ViewSheet and ViewStore is shown in figure
19.4, modified so that upper case letters indicate the minimum
abbreviation for each command. Most commands can be abbreviated,
and abbreviations shouldn't be followed by a dot.

```
ViewSheet              ViewStore

Create n x y           LF file
Headings on/off        LIst
Load file              Load file
LW file                Mode n
Mode n                 NEW
NAme (file)            PREfix (letter (string))
NEW                    PRINTEr file
PC                     SF file
Print                  Utility file
PRINTEr (file)
PROtect on/off
*RC file
*READ file slot (T)
Save (file)
SCreen
SW file
```

Figure 19.4. Summary of ViewSheet and ViewStore commands.

APPENDICES

Appendix A
ViewSheet Quick Reference

ViewSheet Command Mode

The following commands can be used in ViewSheet Command mode. Minimum abbreviations are shown in brackets - either capitals or lower case can be used.

CREATE <n> <x> <y> Creates a blank link file called V.VSn (for example V.VS17), n must be between one and 255. The file has x columns and y rows. See also the READ and WRITE functions. (C).

ESCAPE Switches to Sheet mode. From Sheet mode, ESCAPE returns to Command mode.

HEADINGS {ON,OFF} Row headings and column headings can be switched on or off. Current heading status is shown with just HEADINGS. (H).

LOAD <filename> Loads the model file, including the printer and screen window definitions saved with the model. The file must have been saved with the SAVE command. (L).

LW <filename> Loads the window file. The file must have been saved with the SW command.

MODE <n> Changes the screen display to mode n. Any screen window definitions are lost, but printer windows are retained. (M).

NAME <filename> Sets the default filename, which can then be used with the SAVE command. (NA).

NEW	Clears the model and window definitions from memory, leaving a blank sheet. There is no OLD command.
PC	Prints the name of each occupied cell together with its contents. Useful only for debugging a model, or in creating a contents file for OverView.
PRINT	Prints the model, using any printer window definitions if they are set. (*P*).
PRINTER <filename>	Loads a VIEW family printer driver. The driver is used with the PC and PRINT commands. Individual printer windows can be printed using underline and bold effects. (*PRINTE*).
PROTECT {ON,OFF}	PROTECT disables or re-enables the protection preventing rows and columns being deleted or altered. (*PRO*).
SAVE <filename>	Saves the model and window definitions in the specified file. If no filename is given, then the default name set with the NAME command is used. (*S*).
SCREEN	Displays the model on screen, using the printer windows. Useful for previewing a model before printing. (*SC*).
SW <filename>	Saves the printer and screen window definitions in the file.
*SHEET	Selects and loads ViewSheet from another application.

All other star commands can be used too, though some may corrupt the model in memory. These dangerous commands include: *BACKUP, *COMPACT, *COPY, *FORMAT, *LOAD, *SRLOAD (using Q) and *SRSAVE (using Q). Other star commands like *BASIC and *WORD also lose the model in memory because they select another application.

ViewSheet Sheet Mode

The following functions can be used in ViewSheet Sheet mode. The values n and a represent expressions, which can themselves contain other functions, cell references and the like, as well as constants.

ABS (n)
The absolute value of n. ABS (n) is always positive, even if n is negative.

ACS (n)
Arc-cosine. Gives the angle of which n is the cosine, in radians.

ASN (n)
Arcsine. Returns the angle in radians of which n is the sine.

ATN (n)
The arctangent of n gives the angle of which n is the tangent.

AVERAGE (list)
Gives the mean (average) value in the list. List items may be constants, cell references, ranges of cells or other functions, separated by commas. A range is taken as several separate cells, and blank cells are counted as zero values in averaging.

CHOOSE (n,list)
Chooses the nth value in the list. The list may contain constants, cell references or other functions. A range in the list such as B5B8 is taken as a single item, the sum of the four cells, not as four separate items. CHOOSE is only useful when n is a cell reference.

COL
The number of the current column. Column A is 1, column Z is 26, column IU is 255.

COS (a)
The cosine of a. Angle a must be in radians.

311

DEG (a)	Converts angle a from radians to degrees.
EXP (n)	Exponent of n. EXP is the inverse of LN; it converts from a natural logarithm back to a normal number.
IF (cond,out1,out2)	IF returns the value of outcome1 when the condition is true, otherwise it returns outcome2. The condition can be an expresson like 'A5>10', or an ordinary number. ViewSheet takes the number zero to be FALSE, all other numbers to be TRUE. IF can be nested to simulate AND and OR.
INT (n)	The whole number part of n.
LN (n)	Gives the natural (base e) logarithm of n. See EXP.
LOG (n)	Gives the ordinary (base 10) logarithm of n. There is no anti-log function; the function '10^n' should be used.
LOOKUP (n,cr,rr)	Searches the compare range (cr) for the value n. When n is found, LOOKUP returns the matching value in the result range (rr). Used for things like price lists and looking up values in tables.
MAX (list)	The highest value in the list. The list may be made up of constants, cell references, other functions or ranges, separated by commas. A range is taken as separate cells, not as the sum of the range.
MIN (list)	The lowest value of the items in the list, as with MAX.
PI	3.141596253 (π)
RAD (a)	Converts the angle from degrees to radians. 'COS (RAD (45))' is the cosine of 45 degrees - remember ViewSheet

	trigonometry functions require all angles in radians.
READ (n,x,y)	Reads the value from column x, row y of the link file V.VSn. See WRITE.
ROW	The number of the current row
SGN (n)	gives zero if n is zero, one if n is positive, and minus one if n is negative.
SIN (a)	The sine of the angle a (in radians).
SQR (n)	Square root of n (n must be positive).
TAN (a)	Tangent of a. The angle must be in radians.
WRITE (n,x,y,z)	Writes the value z to the link file V.VSn, in column x, row y.

The following mathematical and conditional operators can be used in expressions or conditions. They are listed in order of their precedence:

group 1	-, ()	unary minus, brackets
group 2	^	power
group 3	*, /	multiply, divide
group 4	+, -	add, subtract
group 5	=, <>	equal, not equal >, >=greater, greater or equal = <, <=less, less or equal

The following can be used in window definitions:

Format (Fmt) F, Dn	Floating or fixed decimal point. With D, n gives the number of decimal places to be displayed.

R, L	Right or left justification of numbers. Text labels can only be changed using SHIFT-f8, the JUSTIFY LABEL function.
M, B	Negative numbers shown with a minus sign or within brackets. M is more familiar, but B is often used for financial calculations.
Options (Opt) C	Bar chart shows cell contents as row of asterisks.
H	Horizontal scrolling linked to other windows (screen windows only).
O	Switch window off.
S	Switch left side border off.
T	Switch top border off.
V	Vertical scrolling linked to other windows (screen windows only).
1, 2	Highlight one or two, underline or bold (printer window only).

ViewSheet Error Messages

ViewSheet may produce the following error messages. These appear normally in Command mode, or in the status area at the top-left hand corner of the screen in Sheet mode.

Bad file	The file is not a ViewSheet file. LOAD can only load files saved with the SAVE command, and LW only loads SW files.
Bad heading	The heading already exists elsewhere.
Brackets	Brackets must be matched in pairs in expressions.

Command?	An invalid command in Command mode.
Divide by 0	The expression involves dividing by zero.
Edge	The model may extend only to row 255, column IU. Replication would involve cells beyond the limit.
File not found	Check the spelling of the filename, and check the file exists with *CAT.
FS error	Some filing system problem prevents WRITE or READ working.
Log range	Logarithms of negative numbers aren't possible.
LOOKUP	ViewSheet failed to find a value in the compare range. Something must match the selection value exactly. Alternatively, the result range is shorter than the compare range.
Memory	The model has run out of memory.
No file	The link file can't be found. Check that it is on the disc, with *CAT V; remember it must be in directory V.
Not enough memory	The model is too big for the screen mode.
No sheet	Probably the sheet has been corrupted by an unsafe * command. Type NEW and reload the model.
Out of range	The number can't be larger than 255. Alternatively, the link file has fewer rows and columns than specified in READ or WRITE.
Overflow	The expression is too complex.

Propagated	Cell reference to a cell containing an error.
Protected	Protection must be switched off before a whole row or column can be deleted, or before a cell in a protected row or column can be edited.
Range	Something wrong with a range specification.
Syntax?	A valid command but with the wrong syntax in Command mode.
Too big	The expression uses a number that is more than 1.7E38, the largest number ViewSheet can use.
Too few arguments	Not enough information supplied to READ, WRITE or the conditional functions IF, CHOOSE and LOOKUP.
Too long	The longest line ViewSheet can accept is 191 characters.
Too many files	Only five link files can be used with one model, or 10 with ADFS.
-ve root	SQR must have a positive number.
%	The value is too large to show in the column width using the current format. It's shown in full in the status area when the cell cursor is on the cell.
?Error	Error in the cell contents. The full error message appears in the status area when the cell cursor is on the cell.

Getting More Memory

Running out of memory with ViewSheet is a serious problem, as there is usually no easy way to simplify an existing model. There are two

solutions - splitting the model in two, or fitting more memory to the computer. Splitting the model will usually involve a link file to carry values from one half to the other. This is really only feasible if there is a natural 'break' in the spreadsheet, say between data processing and analysis phases of the model.

Finding more memory may be the only option. The easiest way is to change to a less memory-hungry mode, perhaps mode 7, but this gives a very poor sheet display. With the Master series or the BBC B+, ensure the shadow screen is in use by selecting, say, mode 131 or typing:

```
*SHADOW
MODE 3
```

This can gain back about 20k of memory for sheets using mode 0 or 16k extra in mode 3. Fitting a 20k or 32k shadow RAM board to a BBC B gains a similar amount. The units from Watford Electronics and Aries are recommended.

The only other way the size of the model can be increased is by using a 6502 second processor or Turbo co-processor, but this adds very little memory if shadow RAM is already in use, typically only about 4k. There is no Hi-ViewSheet. However, a Second Processor speeds up recalculation of large spreadsheets significantly.

Appendix B
ViewStore Quick Reference
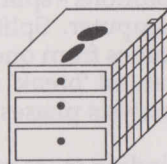
ViewStore Command Mode

The following commands can be used in ViewStore Command mode. Minimum abbreviations are shown in brackets—either capitals or lower case can be used.

ESCAPE	Switches to either Card or Spreadsheet mode, providing a database is loaded. From either data mode, ESCAPE returns to Command mode and saves any changes made to the database.
LF <name>	Loads the specified new format for the database. The name can be a complete filename, or a part-name to which the F prefix is added automatically.
LIST	Lists on screen the fieldnames used by the database. (*LI*).
LOAD <name> {<name>}	Loads the database. A single name can specify both data and format files using the D and F prefixes. For example, 'LOAD CREDIT' could load :0.D.CREDIT and :2.F.CREDIT. Alternatively, separate complete names can be used for the files. (*L*).
MODE <n>	Changes the screen mode. (*M*).
NEW	Restarts ViewStore. There is never any need to save data before typing NEW.

PREFIX {<l> <prefix>} Used to set the five prefix strings - D, F, I, S and U. These prefixes are added to filenames before loading and saving files. PREFIX without parameters displays the five current prefixes. (*PRE*).

PRINTER <filename> Loads a printer driver. The driver can be used to allow the use of highlights or special characters in reports, for example. (*PRINTE*).

SF <name> Saves the current database format in the specified file, using the F prefix if appropriate.

UTILITY <name> Runs a utility program. The U prefix is added automatically to the name given.

UTILITY CONVERT Utility to change the order of fields in the database, insert more space for records, purge deleted records, etc. (*U CONVERT*).

UTILITY INDEX Builds or rebuilds the index file on some specified field. (*U INDEX*).

UTILITY LINK Writes data from the database records to a ViewSheet link file. (*U LINK*).

UTILITY MACRO Writes data from the database records to a VIEW macro file. (*U MACRO*).

UTILITY REPORT Prints or displays a report, if necessary using a report definition file. (*U REPORT*).

UTILITY SELECT Makes a selection file containing just a part of the overall database. The selection file can then be sorted and used with most of the other utilities. (*U SELECT*).

UTILITY SETUP	Utility to set up a fresh database, or create a new report definition. (*U SETUP*).
*STORE	Selects ViewStore from another application.

All other star commands can be used too, though some may corrupt the format file in memory, and reloading the database may be necessary. These dangerous commands include: *BACKUP, *COMPACT, *COPY, *FORMAT, *LOAD, *SRLOAD (using Q) and *SRSAVE (using Q). However, the database should not be vulnerable, as all the data is saved on disc before returning to Command mode.

ViewStore Error Messages

ViewStore may produce the following error messages, either in Command mode, or Data mode. In Card or Spreadsheet mode, the error message is displayed in the status area of the screen at the upper left corner.

Bad date	The date doesn't match the dd/mm/yy form required (mm/dd/yy for American dates), or the day of the month doesn't exist (for example 31/9/87).
Bad directory	ViewStore expects files to be in the correct directory, for example format files in the F directory. Check the prefixes are correct for each type of file.
Bad drive	Usually this means a dot left off the end of a prefix string, eg, 'PREFIX D :2'.
Bad expression	An invalid expression in a report definition. Check the list of database fieldnames, and also that there are delimiters around all names containing wildcards in the report field list.
Bad field	The field doesn't exist. Try LIST to check all the fieldnames.

Bad FS	ViewStore doesn't work with tape.
Bad macro	Only two letter names are possible.
Bad mode	Mode requires a sensible number.
Bad name	The filename isn't valid. Check the prefixes all end in a dot.
Bad pointer	The index file needs rebuilding.
Bad prefix	Only D, F, I, S and U prefixes can be set.
Bad record size	Record size requires a sensible number, possibly with a +.
Bad register	Only A: to Z: are available. R: and P: shouldn't be used as they are reserved for the record and page numbers.
Bad selection	Selections must consist of fieldnames, operators and values. The valid operators include brackets, AND and OR.
Bad string	The index name in the database format is invalid.
Brackets	Brackets must be balanced in expressions.
Can't extend	A file has grown too big.
Channel	Go to Command mode, type NEW and reload the database.
Data screen only	Spreadsheet or Card mode only.
Disc full	A file has grown too big.
Divide by 0	The REPORT definition contains an expression that involves dividing by zero. This is commonly due to dividing by a numeric field which is left blank for some records.

321

Editing no file	Load a database before leaving Command mode.
End	The beginning or the end of the database.
Escape	Escape.
Field is not numeric	LINK files can only carry numerical information.
Field not found	The field doesn't exist. Try LIST to check the current fieldnames.
File exists	Files of the same name already exist. Either delete the existing format and data files, or pick a new name for the new database to setup.
File more than ...	The link file is beyond the maximum size possible.
File not wide enough	Not enough columns in the link file for the number of fields used.
Filename not found	ViewStore can't find the file. This may be because it doesn't exist, or because the PREFIX is causing ViewStore to look in the wrong directory or on the wrong drive.
File not open	A file can't be opened. Return to Command mode and reload the database.
File open	A file has been left open. Try typing NEW or *CLOSE, then reload the database.
Fixed format	The report definition format file is normally un-editable.
Index: bad filename	The index filename in the record format is invalid. Also check the I prefix is correct.

Index: file not open The index file doesn't exist, or the I prefix is wrong.

Index: can't extend The index file doesn't have any room to grow on the disc. It must be deleted then rebuilt using the INDEX utility.

Index: not found ViewStore can't find the index file. Check the name in the database.

Key too long The sorting criteria are too complex.

Keysize should be ... Use a reasonable key width for the index.

Locked The database and format files should not be locked. Unlock them with *ACCESS, and try again.

Limit error The value is outside the specified limits.

Maximum is 4/9 indexes There can only be four updateable index files with DFS and Network, or nine with ADFS. Remember some can be made read-only by putting R instead of Y in the record format.

Memory full The utility needs more memory. Change to a lower resolution mode; in most cases mode 6 is adequate, extreme cases may need mode 7.

Mistake An invalid command in Command mode.

No data Probably the database in memory has been corrupted by an unsafe star command - type NEW and reload the database.

No database loaded All the utilities need the database loaded first (except for SETUP).

No end marker The database file is corrupt. Make a copy of the back-up file and use that. If there is

	no backup, then the IMPORT utility may be able to read most or all of the file.
No fields	At least one field must have a name before using Card mode.
No fields found	The printer width must be larger than the width of the first field. Either reduce the width of the field, or specify a wider printer width.
No index	Y or R (for updateable or read-only), must be set in the database format for the field.
No index field	There are no indexes - the database can only be sorted by entry.
No records found	The selection criteria are not met by any fields in the database.
Not enough fields	There are too few patterns in Half1 and Half2 to match the number of items in the field list.
Not numeric	Numeric fields can only hold numbers.
Overflow	A number or result in an expression is either too big or too small.
Read error	The file is corrupt. Make a copy of the back-up of the file, and use that. If using a floppy disc, treat it as suspect, and copy the database onto a more reliable disc.
Record too big	ViewStore can't read the next record because there isn't enough memory left. Change to a more frugal screen mode to increase the free space, or reduce the capacity figure in the database header.
Sheet display	Card mode only.

Stack overflow

The expression is too complex. This also occurs with the INDEX utility, when building an index for a field which is already sorted. Make sure the selection file is unsorted before using it with INDEX.

Too long

A filename can't be more than 12 characters. This can be a problem with ADFS - either use a different prefix, or use wildcards in the filename, for example 'VST*.CREDIT'.

Too many files

There can only be four updateable index files with DFS and network, or nine with ADFS.

Too many places

The precision is limited to the number of decimal places specified in the record format.

Type mismatch

Arithmetic can only be performed on numeric data.

Value not in list

The value is not in the value list.

Can't Extend

With DFS, when a data or index file grows too big, space must be cleared for it to grow. This can be done with the following procedure: return to ViewStore Command mode, and copy the file on to a spare disc. Delete it from the original disc, then compact that disc. Now copy the file back from the spare disc.

This places the file at the end of the disc, where is has the maximum space to grow. However, having two growing files on the same drive is very tedious. When setting up the database, always reserve lots of room for the index files, as they will always be on the same drive. If at all possible, arrange the prefixes so that the data file is on a drive of its own.

Extra Memory

ViewStore does not need vast amounts of space to keep the entire data file in memory, but it does need to hold the entire format file, plus any utility program, plus one or more records. With the REPORT utility, it may need to hold the report definition file in memory too. With large records, and complex formats, this may strain the memory capacity of a BBC model B in high-resolution modes (0 or 3). Switching to mode 6 before using a utility program is usually enough, but if 'Record too big' errors recur, extra memory may be needed.

With the Master series, working in a shadow mode will free an extra 16k of memory in mode 3. Using mode 131, or typing *SHADOW before changing mode accomplishes this. On a BBC B, a shadow RAM board can offer the same. Using a 6502 second processor or Turbo Co-processor offers enough extra memory for all practical purposes, a further 4k or so in machines with shadow RAM, or about 24k in an original model B. The extra memory often allows the capacity figure to be increased substantially, perhaps to its maximum, 50. This speeds things up because more records are held in memory at one time.

The second processor doesn't speed up ViewStore very much, because its speed is limited by the speed of getting data from disc. Using ADFS and a hard disc makes a much greater improvement in ViewStore's performance, because the data transfer is much speedier.

Appendix C
ViewPlot Quick Reference
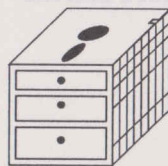
Elements of the ViewPlot system may produce the following error messages. They may be displayed by the data editor, the format editor, the graph plotter or the pattern editor modules.

Data set not found

The data editor LOAD command can't load a file that doesn't exist. Make sure the correct data disc is in the drive. In graph plotter, check filename in format file is correct.

D/F file not found

The graph plotter can't find the specified file. Check the correct disc is in the drive and the filename is correct.

File not a data set

The file must have been saved by the data editor.

File not a pattern file

File must have been saved by the pattern editor.

File not a format file

File must have been saved by the format editor.

Format file not found

The format file doesn't exist. Make sure the correct disc is in the drive, and that the filename given is correct.

GXR active

The Graphics Extension ROM is active on a model B or B+. Press BREAK and start again.

Link file not found

ViewPlot can't find the specified V.VS link file.

Negative Y data in pie

Pie charts can only plot positive values.

Not a link file

The file is not a valid link file. Link files can be created only by ViewSheet's CREATE command, by the ViewStore LINK utility, and by special programs like the enhanced RWLINK utility on the disc accompanying this book.

Pattern file not found

Pattern editor can't find the pattern file - make sure the filename is correct and the right disc is in the drive.

Pie sum too small

Y values are too small for accuracy.

Spool file not found

Check the filename is correct, the same given with *SPOOL, *PREPARE or to the ASCII spooler.

This is not a D/F file

The graph plotter can use only data and format files saved by the data editor and the format editor.

This is not a F file

The next chained format is not a valid format file.

Appendix D
OverView Star Commands
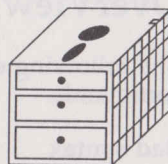
The following star commands can be used if OverView is present.

*KEEP {ON,OFF,RAM,QUIET, \<prefix\>}

> Switches the Keeper on or off, using disc storage for the context files. It can also be used to set a prefix for the context files. *KEEP RAM makes OverView use the four banks of sideways RAM in a Master 128 to store two complete contexts, reverting to disc for subsequent files. *KEEP QUIET temporarily disables the Keeper. *KEEP alone displays the current Keeper status.

*RC \<filename\>

> Reads a ViewSheet 'contents' file into another ViewSheet model. This way it allows transfer of large amounts of data from one model to another.

*READ \<filename\> \<cell\> {T}

> Reads a correctly formatted text file into a ViewSheet model, beginning at the specified cell. The text could be produced by VIEW or even a ViewStore report.

*WIDE {ON,OFF}

> Switches the wide screen system on or off. This gives 106 or 53 characters in screen modes normally showing only 80 or 40 characters. The wide screen works only in two-colour modes (0, 3, 4 or 6 plus their shadow equivalents).

329

OverView Errors

The following error messages are associated with the OverView star commands.

Bad syntax   Wrong syntax for a OverView command.

No page   OverView requires a Master series micro.

Not found   OverView can't open the file. Use *CAT to check the file exists, and *KEEP to check the context file prefix.

Too much to keep   With a Master Turbo or 6502 second processor, a VIEW document longer than 30k can't be saved in a context file. Save it normally and use *KEEP QUIET to disable the Keeper.

Appendix E
ViewSheet and ViewStore
Programs Disc
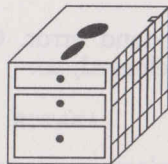
A disc of software is available to accompany this book from Dabs Press.
It contains all the programs listed in this book, plus a number of new
utilities for ViewSheet and ViewStore users and many of the major
example models and databases:

  VSXFER and ASCII spoolers for the VIEW family
  JOIN, PURGE and MULTI-MACRO utilities for ViewStore
  PAGE on-screen page previewer
  SIDEPRT prints large spreadsheets sideways
  RWLINK enhanced Basic link file handler
  DECODE exec file decoder plus selection of exec files
  *CLOSE utility
  *BATCH utility
  !BOOT files for ViewStore and ViewSheet
  BEACH and INVOICE models
  BIBLIOG and MEMBERS databases

*CLOSE is a utility for DFS users with BBC model B micros. Sometimes,
files are left open, for example when BREAK is pressed. Files left open
can't be deleted or modified. The BBC B+ and Master series micros
provide a *CLOSE command to shut all open files. This utility does the
same on a normal BBC B.

*BATCH makes exec files more intelligent. It allows exec files to contain
patterns, %1 to %9. These can be replaced by arguments as the file is
being read. So an exec file called 'RELEASE' could include:

```
*COPY :0 :1 %1
*DELETE :0.%1
*COMPACT :0
*COPY :1 :0 %1
```

This exec file carries out the procedure described in Appendix Two for
releasing extra room on a disc to allow a file to grow after a 'Can't

extend' error. Clearly, %1 represents the name file of the file to be released, so:

```
*BATCH  RELEASE  D.CUSTOM
```

releases a file called 'D.CUSTOM'. Any file can be released simply by supplying its name in the *BATCH command. This way, exec files can become totally general purpose.

The disc is available for the BBC B, B+ and Master 128 micros on a 5.25-inch 40-track disc in DFS format. If you have an 80-track disc drive, a copy program converts the disc to 80-track format. The software is also available on an ADFS 3.5-inch disc for Electron, Master Compact, Archimedes and other ADFS users. The disc is unprotected, and the files can easily be transferred to hard disc or network fileserver. It is supplied in a wallet, complete with full instuctions for a cost of £7.95 (5.25-inch) or £9.95 (3.25-inch), inclusive of VAT, postage and packing. Please add £2.00 surface, or £6.00 air mail for orders outside the UK or BFPO.

To obtain your copy of the disc, fill in the order form below, and a copy, or just write a letter. Send it and a cheque, postal order, sterling money order, (or official/company/government purchase order) or your Access or Visa number to Dabs Press at the address on page ii. Telephone credit card orders are also accepted.

Please rush me a copy of the ViewSheet and ViewStore: A Dabhand Guide Programs Disc. I enclose £_____ for a 5.25" / 3.5" version.

Name..

Address...

....

....

Tick here if you require a VAT receipt.................

Appendix F
Dabhand Guides Guide
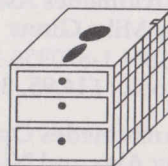
The following books and software packs covering the BBC and Master series micros are published or planned for 1988. Leaflets are available on all products which go into considerably more detail than space here permits. Publication dates and contents may change. All quoted prices are inclusive of VAT (on software—books are zero-rated), and UK postage and packing. (Abroad add £2 or £10 airmail). All are available from your local dealer or in case of difficulty direct from Dabs Press.

VIEW: A Dabhand Guide by Bruce Smith
ISBN 1-870336-00-3 Available Now. 256pp. Book: £12.95. Disc: 5.25in £7.95, 3.5in £9.95. Book and disc together £17.95 (ADFS £19.95)

This is the most comprehensive tutorial and reference guide written about using the VIEW wordprocessor. A suite of VIEW utility programs are provided including VIEW Manager, an extendable front end. An excellent companion to *ViewSheet and ViewStore: A Dabhand Guide.*

Master Operating System: A Dabhand Guide by David Atherton
ISBN 1-870336-01-1 Available Now. 272pp. Book £12.95 5.25" disc £7.95 3.5" disc £9.95 Book/disc £17.95 (3.5" £19.95)

Acclaimed reference guide for programmers and users of the BBC B+ and Master Series micros. Contains a wealth of information on the Operating System, including all star commands, OSBYTE and OSWORD calls, the Tube, filing systems, the non-volatile RAM, differences between all BBC machines and much much more.

Master 512: A Dabhand Guide by Chris Snee
ISBN 1-870336-14-3
Publication: May 1988. 256 pages approx. Book: £14.95 Disc £9.95

Contains all that you are likely to want to know about your Master 512 serving as a tutorial and reference guide.

Bumper Assembler Bundle by Bruce Smith
Publication : Available Now,2 books, 2 discs and booklet: Just £9.95

Five-part package of assembly language materials at less than a third of their normal price. Full details of package on request.

Archimedes Assembly Language: A Dabhand Guide
by Mike Ginns
ISBN 1-870336-20-8 Publication: June 1988 350pp approx.
Book : £14.95, 3.5" disc: £9.95 Book/disc £21.95

Archimedes Operating System: A Dabhand Guide
By Alex and Nick van Someren
ISBN: 1-870336-48-8 (Programs disc 1-870336-49-6) Available July 1988
250 pages approx. Price: £14.95. 3.5" disc: £9.95 Book/disc £21.95

C : A Dabhand Guide by Mark Burgess
ISBN 1-870336-16-X (Programs Disc 1-870336-22-4) Available April
1988 500pp approx. Price: £14.95. 3.5" disc: £9.95 Book/disc £21.95

With sections specific to Archimedes and Master Cs.

BBC and Master Software Packs

View Executive by Graham Bell
Available September 1988 Software pack on disc. Price £19.95.
(formerly View Family Utilities)

This package for VIEW, ViewSheet and ViewStore users contains a host
of useful utilities designed to enhance the power of the suite. Contains
two discs and 100-page manual.

HyperDriver by Robin Burton
Available now ROM £29.95. Sideways RAM version £24.95

The ultimate printer ROM including: on-screen preview, CRT graphics,
NLQ font, user definable macros, and 80 * commands to control your
printer attributes in plain English. Compatible with VIEW family. In-
cludes 100 page manual and demo files.

FingerPrint by David Spencer
Publication : Available Now 5.25" disc £9.95, 3.5" disc £11.95

FingerPrint is a single-step machine code tracing program. It allows
you to step through machine code written by yourself, or part of a com-
mercial program. Includes disassembler/memory editor.

MOS Plus by David Spencer
Available now ROM £12.95, Disc for Sideways RAM £7.95 (3.5" £9.95)

The ROM for all Master 128 users (only) providing ADFS *FORMAT, *VERI-
FY , *BACKUP, *CATALL and *EXALL and several new '*' commands.

SideWriter by Mike Ginns
Publication: Available now 5.25" disc £7.95, 3.5" £9.95

For Sideways RAM owners (incl. Masters), a pop-up notepad which can be used from any program.

Conversion Kit by Bruce Smith
ISBN 1-870336-06-2 Available now. 5.25" disc £7.95 / 3.5" £9.95

A disc containing 24 expert routines that you can use in your own programs. Each routine is held within a BASIC and assembler program as a procedure that demonstrates how to use it.

Master Emulation ROM by David Spencer
ISBN 1-870336-23-2 Available now ROM £19.95 (Disc for SRAM £14.95)

This new ROM software is especially for Model B and B+ owners, and provides you with most of the features of the Master 128.

Other Books from Dabs Press:

AmigaDOS: A Dabhand Guide by Mark Burgess
ISBN 1-870336-47-X Publication : July 1988. 300 pp. Price: £14.95

WordStar 1512: A Dabhand Guide by Bruce Smith
Including WordStar Express
ISBN 1-870336-17-8 Publication : June 1988. 280 pp. Book: £12.95. Disc: £7.95 Book and disc £17.95

PCW 9512: A Dabhand Guide by John Atherton
ISBN 1-870336-50-X. Publication : Late 1988. 300 pages approx.

WordPerfect: A Dabhand Guide by Bruce Smith
ISBN 1-870336-53-4. Publication : Early 1989. 350 pages approx.

Ventura Publisher A Dabhand Guide : Simon Williams
ISBN 1-870336-52-6. Publication : Early 1989. 300 pages approx.

PostScript: A Dabhand Guide by Paul Martin
ISBN 1-870336-54-2. Publication : Early 1989. 300 pages approx.

NB: All future publications are in an advanced state of preparation. Content lists serve as a guide but we reserve the right to alter and adapt them without notification. If you'd like more information about our books and software then drop us a line at the address on page ii.

Index

A Dabhand Guide

This book is a complete tutorial and reference guide for the ViewSheet spreadsheet and the ViewStore database manager. It is specifically written to appeal both to the beginner and to the more experienced user, whether you wish to check your bank statement or run a million pound business. Every aspect of setting up and using a database or spreadsheet is described in detail, and numerous examples are provided to guide you. There are also a number of utility programs to help you get more out of the VIEW family, including programs that join two databases together and help transfer spreadsheets into a wordprocessor. OverView and ViewPlot are also examined and explained. The many features of this book include:

● Compatible with BBC B and Master Series
● Usable with DFS, ADFS and network
● Simple spreadsheet and databases
● Absolute and relative replication
● Building an invoice system
● Database design
● Use of SELECT and REPORT
● Using a printer
● Hints and Tips
● OverView and ViewPlot

Graham Bell is Technical Editor of Acorn User magazine, and a respected authority on the BBC Micro. An expert on the VIEW family, he has written numerous articles on the use of VIEW, ViewSheet and ViewStore. Formerly he was a student at Oxford University where he graduated in Geography and spent a further four years undertaking research at Reading University.

£12·95